Al-Azhar University
Faculty of Engineering
Electrical Engineering department
Electronics and Communication

Graduation Project

Design & Verification and FPGA implementation of MIPI-D-PHY Transceiver

Submitted by

Ahmed Mohamed Ali Ahmed Elmasry	104502
Ahmed Mohamed Hamed Almallah	104503
lsayed Talaat Elsayed Tmmam	104504
Mohamed Emam Hussien Abd Elsalam	104513
Mohamed Mansour Osman Ahmed	104519
Yossif Abdelfttah Ibraheem Uonis	104526

Supervised by

Dr. Abdallah Hassan

Sponsored by Si-Vision Inc.

July 2025

DECLARATION

We hereby certify that this material, which we now submit for assessment in the program of study leading to the award of a Bachelor of Engineering at Al-Azhar University, is entirely our own work. We have exercised reasonable care to ensure that the work is original and, to the best of our knowledge, does not breach any law of copyright. Any work taken from the work of others has been cited and acknowledged within the text of our work.

Signed: by all students.

Name	**Signature**
Ahmed Mohamed Ali Elmasry	
Ahmed Mohamed Almallah	
Elsayed Talaat Elsayed Tmmam	
Mohamed Emam Abd Elsalam	
Mohamed Mansour Osman Ahmed	
Yosif Abdelfattah Ibrahim Uonis	

Date: 12, july 2025.

ACKNOWLEDGEMENT

We are grateful to have accomplished this ramping process and the outcome of this project, which required a lot of guidance and assistance from many people. We are extremely privileged to have had this opportunity. All that we have done is due to such supervision and assistance, and we should not forget to thank them.

We are thankful to **Dr. Abdallah Hassan**, Department of Electronics and Communication Engineering, Al Azhar University, for supervising our project and providing us with the opportunity to learn something new.

We would also like to express our sincere appreciation to **Eng. Ahmed Zaazaa, Eng. Abdulekareem Mohamed, Eng. Ahmed Khalil, Eng. Dina Saad and Eng. Mahmoud Yaser** from **Si-Vision Company** for their continuous support, valuable guidance, and technical insights that contributed significantly to the success of our project.

Finally, we are grateful for all the encouragement and support we received from our families, teaching staff, and colleagues.

ABSTRACT

This document presents the graduation project report prepared by final-year students from the Communications and Electronics Engineering Department at the Faculty of Engineering, Al-Azhar University. The focus of this project is the **design and verification of the MIPI D-PHY**—a high-speed physical layer interface widely used in mobile and imaging applications.

The report outlines the systematic approach taken to implement the D-PHY protocol, beginning with **specification extraction** from the MIPI D-PHY standard. This step ensures that all functional and timing requirements are clearly understood and translated into an accurate RTL design.

The **design phase** details the architectural breakdown of the transmitter (Tx), receiver (Rx), and clock lane subsystems, along with the logic used to support high-speed (HS) and low-power (LP) transmission modes. Special attention is given to the challenges of implementing bidirectional data lanes and handling complex timing requirements.

To ensure correctness and robustness, we adopted a **Universal Verification Methodology (UVM)** using **SystemVerilog (SV)** to create a reusable and scalable verification environment. The **UVM testbench** includes agents, sequencers, drivers, monitors, and a scoreboard to thoroughly test the functionality of the D-PHY design. We created various test scenarios to cover normal operation, boundary conditions, and potential corner cases.

Simulation results are presented to demonstrate the effectiveness of the verification environment. These include waveform snapshots, functional coverage reports, and assertions, showing that the design meets the specified requirements.

Overall, this project provides both a functional MIPI D-PHY implementation and a comprehensive verification solution, highlighting the students' capabilities in both digital design and verification disciplines.

TABLE OF CONTENTS

Contents

CHAPTER 1: INTRODUCTION 11

1.1 INTRODUCTION 11

1.2 D-PHY 11

1.2.1 Introduction 13

1.2.2 Architecture 13

1.2.3 HIGH Frequency Clock Generation 14

1.3 CLOCK LANE 14

1.3.1 High-Speed Clock Transmission 14

1.3.2 Clock Lane Ultra-Low Power State 17

1.4 High-Speed Data Transmission 17

1.4.1 Start of Transmission 18

1.4.2 End of Transmission 19

1.4.3 High-Speed Data Transmission Brust 19

1.4.4 High-Speed Data PPI signals 20

1.4.5 High-Speed Transmit from the Master Side 22

1.4.6 HS Data Reception Example 22

1.5 ESCAPE MODE DATA LANE 23

1.5.1 Remote Triggers 24

1.5.2 Low-Power Data Transmission 24

1.5.3 Low-Power Data Reception 25

1.5.4 Ultra-Low Power State 26

1.6 Bi-directional Data Lane Turnaround 27

1.7 Escape Mode PPI signals 30

CHAPTER 2: DESIGN 37

2.1 MIPI D-PHY CLOCK LANE TRANSMITER TOP LEVEL 38

2.1.1 Block Diagram 38

2.1.2 Interface Table 39

2.1.3 Architecture 39

2.2 MIPI D-PHY CLOCK LANE RECEIVER TOP LEVEL 42

2.2.1 Block Diagram 42

2.2.2 Interface Table 42

2.2.3 Architecture 43

2.3 MIPI D-PHY TRANSMITER LANE TOP LEVEL 46

2.3.1 MIPI D-PHY TRANSMITER CONTROL UNIT 48

2.3.2 MIPI D-PHY HIGH SPEED TRANSMITTER (HS_TX) 50

2.3.2.1 Block Diagram ……..... 50
2.3.2.2 Interface Table ……..... 51
2.3.2.3 Architecture 51
2.3.3 MIPI D-PHY ESCAPE MODE ……..... 54
2.3.3.1 Block Diagram ……..... 54
2.3.3.2 Interface Table ……..... 55
2.3.3.3 Architecture 55
2.4 MIPI D-PHY RECIEVER DATA LANE TOP LEVEL ……..... 59
1. Initialization ….... 60
2. Manage high speed ……..... 61
3. Manage Escape mode operation 62
4. Manage turn around operation 63
2.4.1 MIPI D-PHY HIGH SPEED RECIEVER (HS_RX) 64
2.4.1.1 Block Diagram ……..... 64
2.4.1.2 Interface Table ……..... 65
2.4.1.3 Architecture 66
2.4.2 MIPI D-PHY ESCAPE MODE ……..……..... 68
2.4.2.1 Block Diagram ……..... 69
2.4.2.2 Interface Table ……..... 69
2.4.2.3 Architecture 70
CHAPTER 3: VERIFICATION 73
3.1 INTRODUCTION..... 73
3.2 Project Scope and Objectives 74
3.3 The test bench architecture……..... 74
3.4 Verification Methodology……..... 75
3.4.1 UVM-SV…….…… 76
3.4.2 UVM Class Library and Component Architecture……..…… .. 76
3.4.3 UVM Component Hierarchy Environment ……..…… … 77
3.4.4 The Typical UVM Testbench Architecture…….…… 78
3.4.5 UVM Hierarchy……..……..... 85
3.4.6 UVM Dynamic objects Sequence……..…… 86
3.4.6.1 List of RX Verification Sequences……..…… 87
3.4.6.2 List of TX Verification Sequences…….…….. 87
3.4.6.3 List of FULL SYS Verification Sequences…….……..... 89
3.4.6.4 Virtual Sequence…….……. 90
3.4.6.5 configuration…….…….. 90
3.5 DPI-C Integration Between MATLAB Simulink and UVM ……... 91

3.5.1 Introduction: Co-Simulation Between MATLAB and SystemVerilog .. 91
3.5.2 DPI-C Technology Overview……..……..... 93
3.5.3 Integration Workflow in Detail…….…….. 93
3.5.4 Detailed Functional Interaction…….…….. 93
3.5.5 DPI-C Wrapper Design…….…….. 94
3.5.6 UVM Side Usage…… 94
3.5.7 Advantages and Verification Benefits…… 94
3.5.8 Performance and Limitations…….. 95
3.5.9 Future Extensions……... 95
3.5.10 Conclusion…….... 95
3.6 Verification Plans……..... 96
3.6.1 Verification Plan RX……..... 96
3.6.2 Verification Plan TX……. 100
3.6.3 Verification Plan FULL RTL…… 106
3.7 VERIFICATION SIMULATION RESULTS……..... 110
3.7.1 RX Assertions……... 110
3.7.2 TX Assertions……... 113
3.7.3 Functional Coverage……. 114
3.7.4 Code Coverage……. 116
CHAPTER 4: RESULTS 118
4.1 BEHAVIORAL SIMULATION RESULTS 118
4.1.1 High Speed Mode Proper Operation 118
4.1.2 Escape Mode Proper Operation 122
1. Low power data transmission Mode (Escape Mode) 122
2. Trigger Mode (Escape Mode) 123
3. Ultra Low Power State Mode (Escape Mode) 124
4.1.3 Turnaround (LP Mode Bidirectional Control) 125
4.2 SYNTHESIS RESULTS 126
4.2.1 Master Side..... 126
4.2.1.1 Constraints..... 126
4.2.1.2 Block Diagram Resulted..... 126
4.2.1.3 Timing Reports Resulted..... 127
4.2.1.4 Power Reports Resulted..... 128
4.2.1.5 Area Reports Resulted..... 128
4.2.2 Slave Side..... 129
4.2.2.1 Constraints 129
4.2.2.2 Block Diagram Resulted..... 129

4.2.2.3 Timing Reports Resulted..... 130
4.2.2.4 Power Reports Resulted..... 131
4.2.2.5 Area Reports Resulted..... 131
4.3 LINTING RESULTS..... 131
4.4 IP RESULTS..... 132
4.4.1 Result Timing for MIPI-D PHY IP using VIVADO..... 132
4.4.2 Power Reports Resulted of IP 132
4.4.3 Utilization of IP..... 132
4.5 FPGA IMPLMENTATION..... 133
4.5.1 Blocks Explanation..... 134
4.5.2 System Scenarios for MIPI D-PHY Validation 137
1. Automated Transmission, Reception, and Verification using Processor 137
2. Sending a Real Image and displaying it on a Screen 142
CHAPTER 5: REFERENCES 144

LIST OF FIGURES

Figure 1.1: Universal Lane Module Functions 13
Figure 1.2: Switching the Clock Lane between Clock Transmission and Low-Power Mode 15
Figure 1.3: High speed clock transmission state machine 16
Figure 1.4: Clock Lane Ultra-Low Power State Machine 17
Figure 1.5: HS data transmission 19
Figure 1.6: Example High-Speed Transmission from the Master Side..... 22
Figure 1.7: Example High-Speed Transmission ……..... 22
Figure 1.8: Trigger-Reset Command in Escape Mode 23
Figure 1.9: Low-Power Data Transmission 25
Figure 1.10: Two Data Byte Low Power Data Reception …..... 25
Figure 1.11: Escape mode state machine …….... 26
Figure 1.12: Turnaround Procedure …….... 28
Figure 1.13: Turnaround State Machine ……..... 29
Figure 1.14: Example Turn-around Actions Transmit-to-Receive and Back to Transmit..... 30
Figure 2.1: MIPI D PHY TOP SYSTEM..... 37
Figure 2.2 CLK Lane TX block..... 38
Figure 2.3: Components of CLK Lane TX block..... 39
Figure 2.4: CLK Lane FSM …….. 40
Figure 2.5: CLK Lane RX block …… 42
Figure 2.6 Components of CLK Lane RX block ……..… 43
Figure 2.7: CLK Lane RX FSM block..... 44
Figure 2.8: DATA Lane TX block ……... 46
Figure 2.9: DATA Lane TX Main FSM ……... 48
Figure 2.10: DATA Lane TX HS FSM control 48
Figure 2.11: High Speed TX Block..... 50
Figure 2.12: Components of HS Block..... 51
Figure 2.13: HS TX FSM..... 52
Figure 2.14: Escape Block TX..... 54
Figure 2.15: Components of ESC Block..... 55
Figure 2.16: ESC mode LPDT..... 56
Figure 2.17: Esc mode Trigger..... 57
Figure 2.18: Esc mode ULPS Control..... 57
Figure 2.19: MIPI D-PHY Receiver data lane block diagram..... 59
Figure 2.20: Initialization Slave..... 60
Figure 2.21: HS RX Enter control FSM..... 61
Figure 2.22: Escape mode control FSM..... 62

Figure 2.23: Turnaround control FSM…. 63
Figure 2.24: High Speed RX Block Diagram 64
Figure 2.25: Components of HS RX Block 66
Figure 2.26: HS RX FSM 67
Figure 2.27: Escape RX Block 69
Figure 2.28: Components of ESC RX Block 70
Figure 2.29: Control Unit of Escape mode 71
Figure 3.1: Verification flow…….. 73
Figure3.2: Test bench architecture……74
Figure 3.3: Stand-alone testing…….. 74
Figure 3.4: Back-to-back testing…….... 75
Figure 3.5: UVM ENV……..…... 77
Figure 3.6: UVM Sequence Based Stimulus Generation Architecture……..... 78
Figure 3.7: UVM ENV Component……..... 80
Figure 3.8: UVM AGENT ……..... 81
Figure 3.9: Constrained parameters and unconstrained parameters…….. 83
Figure 3.10: UVM hierarchy……..... 85
Figure 3.11: UVM phases hierarchy…… 86
Figure 3.12: Describes the handshake between the driver and the sequence……..... 90
Figure 3.13: MATLAB RX MODELING……..... 91
Figure 3.14: MATLAB TX MODELING……..... 92
Figure 3.15: MATLAB INTEGRATION WITH UVM ENV……..…... 96
Figure 3.16: RX Assertions……..…….……... 113
Figure 3.17: TX Assertions…….…… 113
Figure 3.18: RX Functional Coverage…….……..... 115
Figure 3.19: TX Functional Coverage…….…… 116
Figure 3.20: RXPHY Code Coverage…….…… 116
Figure 3.21: TXPHY Code Coverage…….…… 117
Figure 4.1: Master high speed entry sequence ……..... 119
Figure 4.2: Slave clock lane high speed entry sequence..…….... 119
Figure 4.3: Slave data lane high speed entry sequence ……..... 120
Figure 4.4: Master high speed exit sequence …….…….... 120
Figure 4.5: Slave data lane high speed Exit sequence ……..... 121
Figure 4.6: Slave clock lane high speed Exit sequence …….……..... 121
Figure 4.7: Low power data transmission …...……..... 122
Figure 4.8: trigger operation ……..... 123
Figure 4.9: ULPS Entry ……..... 124

Figure 4.10: ULPS Exit ……..... 124
Figure 4.11: TurnAround operation ……..... 125
Figure 4.12: Block diagram resulted for Master Side ...……..... 126
Figure 4.13: Constraints report for Master Side…….... 127
Figure 4.14: Clocks report for Master Side …...……..... 127
Figure 4.15: Power report for Master Side ……..... 128
Figure 4.16: Area report for Master Side ……..... 128
Figure 4.17: Block diagram resulted for Slave Side ……..... 129
Figure 4.18: Constraints report for Slave Side ……..... 130
Figure 4.19: Clocks report for Slave Side ……..... 130
Figure 4.20: Power report for Slave Side ……..... 131
Figure 4.21: Area report for Slave Side ……..... 131
Figure 4.22: Result of Linting ……..... 131
Figure 4.23: MIPI D PHY SOC Diagram ……..... 133
Figure 4.24: AXI Interconnect ……..... 134
Figure 4.25: AXI Stream IP Core ……..... 135
Figure 4.26: Direct memory access ……..... 135
Figure 4.27: LPDT Testing code ……..... 138
Figure 4.28: Initialization and LPDT Result ...……..... 138
Figure 4.29: Trigger Testing Code ……..... 139
Figure 4.30: Result of trigger ……..... 139
Figure 4.31: Code of ULPS ……..... 140
Figure 4.32: Result of ULPS ……..... 140
Figure 4.33: Code of TurnAround ……..... 140
Figure 4.34: Result of TurnAround ……..... 140
Figure 4.35: Result of testing in reverse direction ……..... 141
Figure 4.36: Original image ……..... 143
Figure 4.37: TFT image ……..... 143

CHAPTER 1: INTRODUCTION

1.1 INTRODUCTION

This project focuses on the design and implementation of a MIPI D-PHY using Verilog as the hardware description language. The main objective is to develop a functional and efficient D-PHY architecture that adheres to the MIPI D-PHY specification, supporting both high-speed and low-power (escape) modes. Our design includes a Clock Lane and a single Data Lane, enabling point-to-point serial communication. The system operates with two primary clock domains: a 200 MHz high-speed clock and a 20 MHz escape clock for low power signaling and control operations. The project emphasizes architectural decisions, module design, and proper synchronization between modes to ensure reliable data transfer across the physical layer. While a verification environment is employed using SystemVerilog UVM, the core emphasis remains on achieving a robust and synthesizable RTL design.

1.2 D-PHY

1.2.1 Introduction

In this section, we are going to talk about D-PHY. This specification provides a flexible, low-cost, high-speed serial interface solution for communication interconnection between components inside a mobile device. Traditionally, these interfaces are CMOS parallel buses with low bitrates and slow edges for EMI reasons. The D-PHY solution enables a significant extension of the interface bandwidth for more advanced applications and can be realized with very low power consumption. Manufacturers employ the D-PHY specification to create products that adhere to MIPI Alliance interface specifications for mobile devices, including interfaces for camera, display, and unified protocol. By standardizing the interface across products from many manufacturers, implementing this specification decreases time-to-market and design costs for mobile devices. Additionally, by putting this specification into practice, richer feature sets that demand high bit rates can be accomplished. Finally, the extendable nature of the MIPI Alliance Specifications simplifies the process of adding new functionality to mobile devices. The D-PHY standard describes a source-synchronous, high-speed, low-power, low-cost PHY best suited for mobile applications.

The primary purpose of this D-PHY specification is to connect camera and display applications to a host processor, although it can also be used for various other purposes. A dual-simplex arrangement of the same PHY type was also envisioned for linkages in a more general communication network. An asymmetrical operation and data rate are provided by a master-slave relationship between the two sides of the link. The design complexity of the link is greatly reduced by its asymmetrical design, where features such as bi-directional and half-duplex operation are unnecessary. This feature is appealing for applications with asymmetrical data traffic needs or where the cost of separate interconnects for a return channel is prohibitive, though it is optional. It saves applications that don't require return traffic or prefer to use physically distinct return communication channels from incurring unnecessary overhead charges. In our design, we only implement the transmitter data as unidirectional, not bidirectional. We design a clock lane and two data lanes. The D-PHY provides a synchronized connection between the master and slave. A typical PHY configuration consists of a clock signal and one or more data signals. The master generates the clock signal, which is unidirectional and terminates at the slave. Depending on the configuration, data signals can be either unidirectional or bidirectional. The link supports a low-power signaling mode for control and a high-speed signaling mode for fast data transmission. An option for low-speed asynchronous data transmission is using a low-power escape mode. High-speed data transmission occurs in bursts with a variable payload size of data bytes. Each lane in the minimum PHY configuration is terminated on both ends and operates with a low-swing differential signal in high-speed mode, utilizing four wires for each data lane in addition to two wires for the clock lane. In low power mode, all wires operate single-ended and non-terminated. For EMI reasons, drivers in this mode must be slew-rate controlled and current limited. The actual maximum bit rate achievable in high-speed mode depends on the efficiency of transmitter, receiver, and connection implementations, thus not explicitly specified in this document. However, the specification aims to support bit rates ranging from 80 to 1000 Mbps per lane. While PHY configurations are not limited to this range, they represent a practical range for targeted applications, with an expected typical deployment achieving approximately 500 Mbps per lane. Increasing the available data capacity of a PHY configuration for a fixed clock frequency is possible by adding more data lanes.

1.2.2 Architecture

A PHY configuration contains a Clock Lane Module and one or more Data Lane Modules. Each of these PHY Lane Modules communicates via two lines to a complementary counterpart on the other side of the Lane Interconnect.

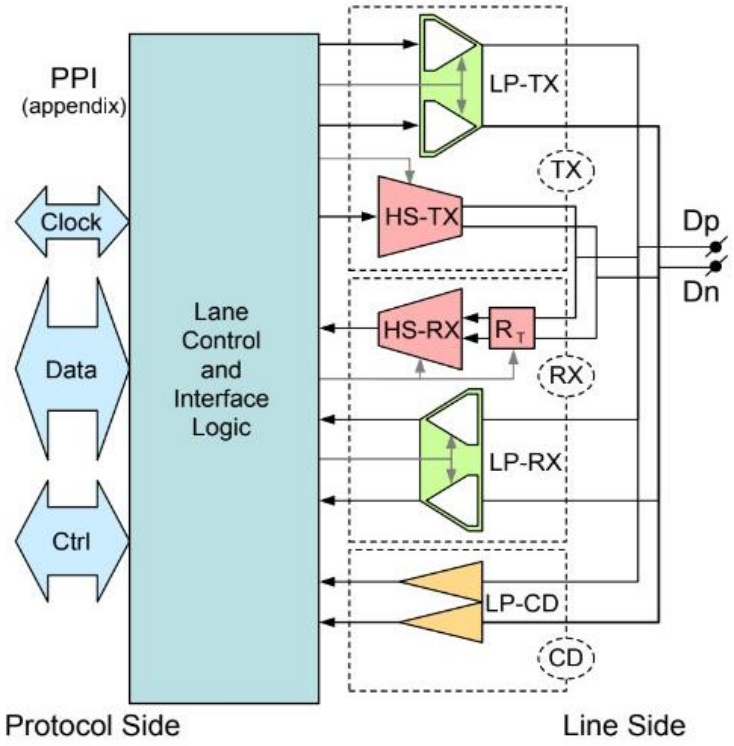


Figure 1.1: Universal Lane Module Functions

Each Lane Module consists of one or more differential High-Speed functions utilizing both interconnect wires simultaneously, one or more single-ended Low-Power functions operating on each of the interconnect wires individually, and control & interface logic. An overview of all functions is shown in Figure 1. High-Speed signals have a low voltage swing, e.g., 200 mV, while Low-Power signals have a larger swing, e.g., 1.2V. High-Speed functions are used for High-Speed Data transmission. The Low-Power functions are mainly used for control but have other optional use cases. The I/O functions are controlled by a Lane Control and Interface Logic block. This block interfaces with the Protocol and determines the global operation of the Lane Module. High-Speed functions include a differential transmitter (HS-TX) and a differential receiver (HS-RX). Low-Power functions include single-ended transmitters (LP-TX), receivers (LP-RX), and Low-Power Contention Detectors (LP-CD). Low-Power functions are always present in pairs, as these are single-ended functions operating on each of the two interconnect wires individually. The presence of High-Speed and Low-Power functions is correlated. That is, if a Lane Module contains an HS TX, it shall also contain an LP-TX. A similar constraint holds for HS-RX and LP-RX.

1.2.3 High Frequency Clock Generation

In many cases a PLL Clock Multiplier is needed for the high frequency clock generation at the Master Side. The D-PHY specification uses an architectural model where a separate Clock Multiplier Unit outside the PHY generates the required high frequency clock signals for the PHY. Whether this Clock Multiplier Unit in practice is integrated inside the PHY is left to the implementer.

1.3 CLOCK LANE

1.3.1 High-Speed Clock Transmission

For High-Speed Data Transmission, the Clock Lane provides a differential, low-swing DDR (half-rate) clock signal from Master to Slave. The clock signal must be in quadrature phase about a forward-moving bit sequence on a data lane and a rising edge in the middle of the first transmitted bit of a burst to function properly. Section 9 contains information on the Data-Clock relationship and timing requirements. Like a Unidirectional Data Lane is a Clock Lane. A Clock Lane sends a High-Speed DDR clock signal rather than data bits due to timing differences. Furthermore, the Low-Power mode capabilities for a Clock Lane and a Data Lane are described differently. A Clock Lane must be unidirectional and exempt from the standard Escape mode features. By using a specific entrance sequence and the LP-Request state, only ULPS shall be supported. High-Speed Clock Transmission must begin in a Stop state and end in. The Clock Lane PPI allows the Protocol to control the Clock Lane module. The Protocol will only end when there are no High-Speed communications going on in any Data Lane, the Clock Lane. If the Clock Lane is in Low Power mode, the start-up time for High-Speed Data Transmission on a Data Lane is prolonged. In that situation, the Transmit Request cannot be processed until the Clock Lane has resumed High-Speed operation. After the final Data Lane enters Low-Power mode and comes to an end with an HS-0 state, the High-Speed Clock signal must continue to operate for a period known as TCLK-POST. Table 1 outlines the steps for putting the Clock Lane in Low-Power mode. Keep in mind that there are always even numbers in Clock Burst. As it begins and ends in an HS-0 condition, it transitions. As a result, it may be inferred that the clock offers transitions to sample with an equal number of bits on any relevant Data Lanes. Clock intervals must be accurate and adhere to HS timing requirements. The table provides the steps to revert the clock lane to high-speed clock transmission.

Figure illustrates the Clock Start and Stop methods.

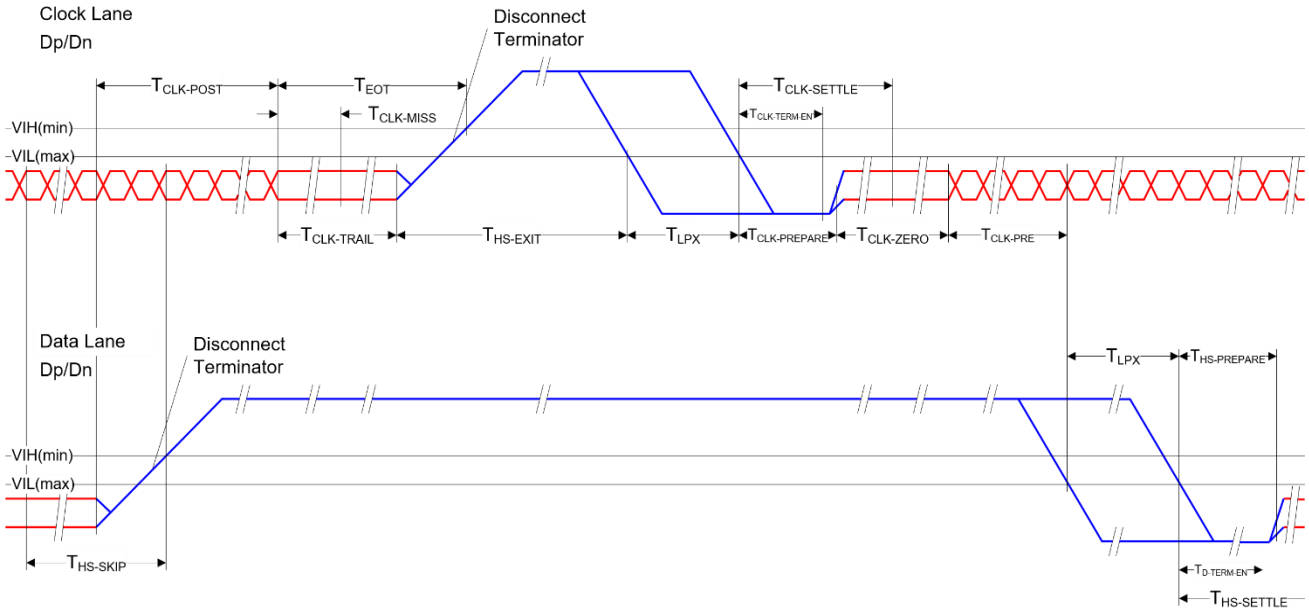


Figure 1.2: Switching the Clock Lane between Clock Transmission and LP Mode

Master Side	**Slave Side**
Drives High-Speed Clock signal (Toggling HS-0/HS-1)	Receives High-Speed Clock signal (Toggling HS-0/HS-1)
Last Data Lane goes into Low Power mode	
Continue to drive High-Speed Clock signal for a period $TCLK_{-POST}$ and ends with HS-0 state	
Drives HS-0 for a time $TCLK_{-TRAIL}$	Detects absence of Clock transitions within a time $TCLK_{-MISS}$, disables HS-RX then waits for a transition to the Stop state
Disables the HS-TX, enables LP-TX, and drivesStop state (LP-11) for a time THS-EXIT	
	Detects the Lines transitions to LP11, disables HS termination, and enters Stop state

Table 1.1: Procedure to Switch Clock Lane to Low-Power Mode

TX Side	RX Side
Drives Stop State (LP-11)	Observes Stop state
Drives HS-Req state (LP-01) for time TLPX	Observes transition from LP-11 to LP-01 on the Lines
Drives Bridge state (LP-00) for time TCLK-PREPAR	Observes transition from LP-01 to LP-00 on the Lines. Enables Line Termination after time T_{CLK_TERMEN}
Enables High-Speed driver and disables Low-Power drivers simultaneously. Drives HS-0 for a time $T_{CLK-ZERO}$.	Enables HS-RX and waits for timer T_{CLK_SETTLE} to expire to neglect transition effects
	Receives HS-signal
Drives the High-Speed Clock signal for period T_{CLK_PRE} before any Data Lane starts up	Receives High-Speed Clock signal

Table 1.2: Procedure to Initiate High-Speed Clock Transmission

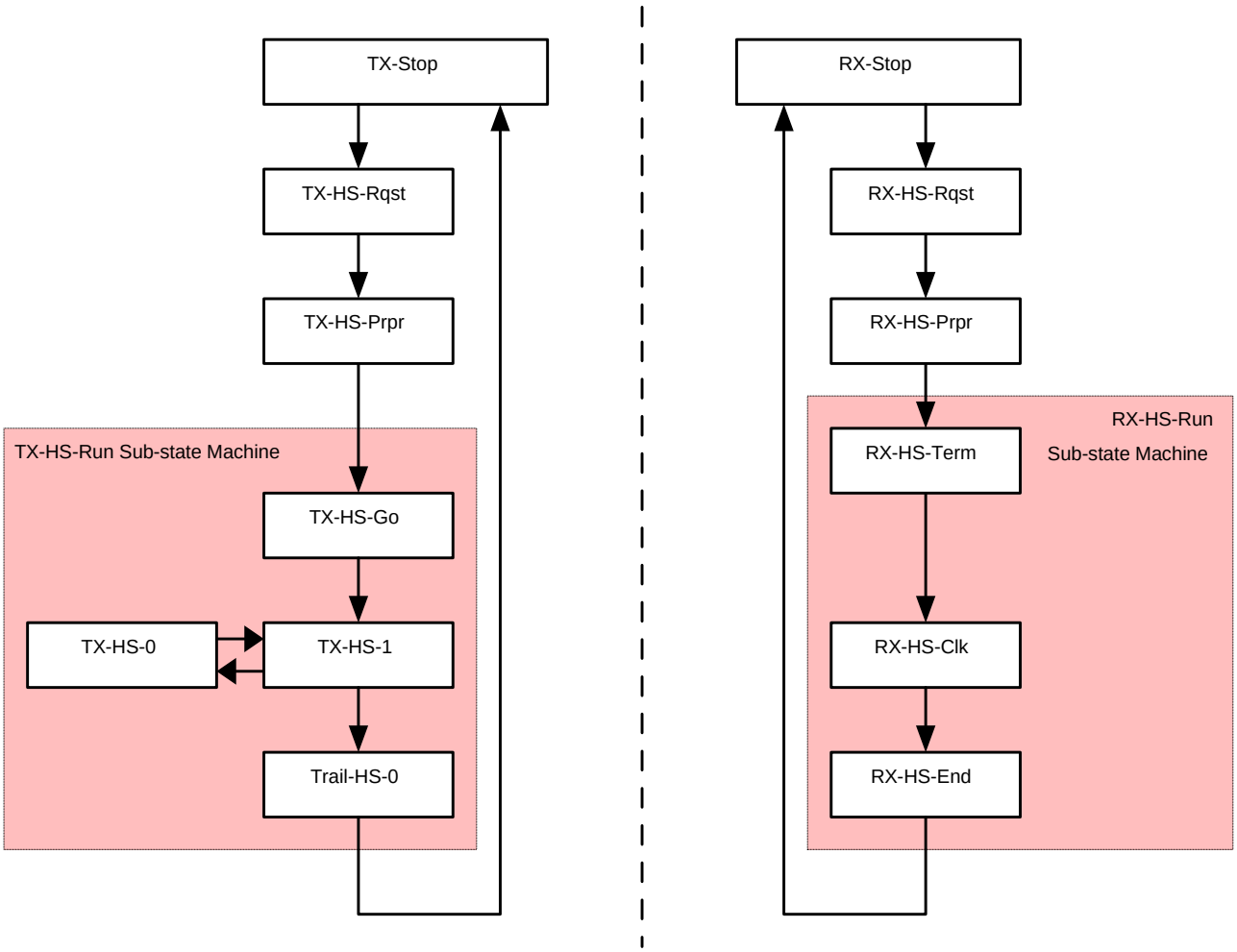


Figure 1.3: High speed clock transmission state machine

1.3.2 Clock Lane Ultra-Low Power State

Although a Clock Lane does not include regular Escape mode, the Clock Lane shall support the Ultra-Low Power State. A Clock Lane shall enter Ultra-Low Power State via a Clock Lane Ultra-Low Power State Entry procedure. In this procedure, starting from Stop State, the transmit side shall drive TX-ULPS-Rqst State (LP-10) and then drive TX-ULPS State (LP-00). After this, the Clock Lane shall enter Ultra-Low Power State. If an error occurs, and an LP-01 or LP-11 is detected immediately after the TX-ULPS-Rqst state, the Ultra-Low Power State Entry procedure shall be aborted, and the receive side shall wait for, or return to, the Stop state, respectively. The receiving PHY shall flag the appearance of ULP State to the receive side Protocol. During this state the Lines are in the ULP State (LP-00). Ultra-Low Power State is excited by means of a Mark1 TX-ULPS- Exit State with a length TWAKEUP followed by a Stop State.

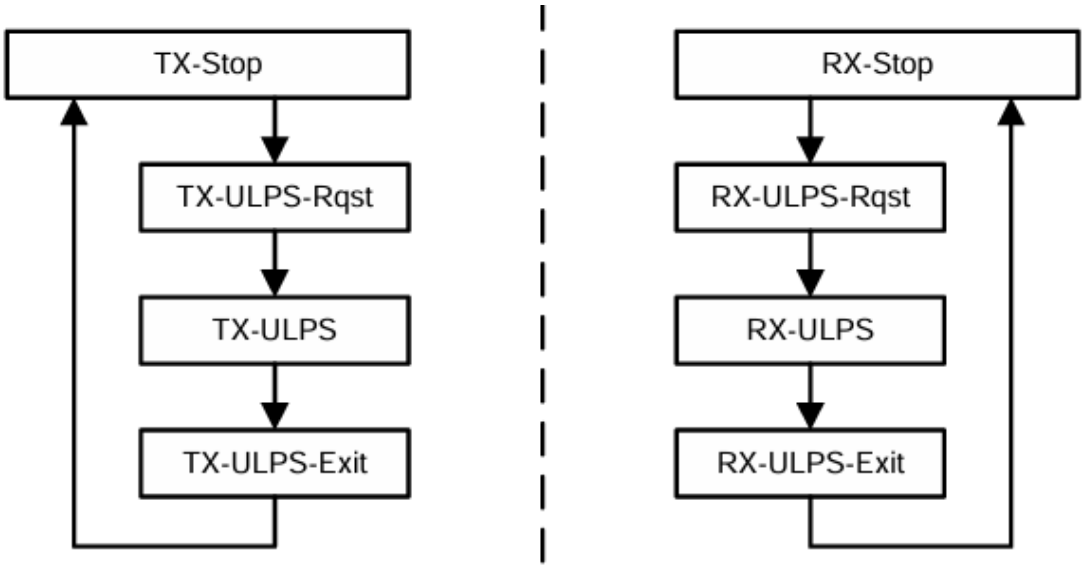


Figure 1.4: Clock Lane Ultra-Low Power State Machine

1.4 High-Speed Data Transmission

High-Speed Data Transmission occurs in bursts. To aid receiver synchronization, data bursts shall be extended on the transmitter side with a leader and trailer sequence and shall be eliminated on the receiver side. These leader and trailer sequences can therefore only be observed on the transmission lines. Transmission starts from, and ends with, a Stop state. During the intermediate time between bursts a Data Lane shall remain in the Stop state unless a Turnaround or Escape request is presented on the Lane. During a HS Data Burst the Clock Lane shall be in High-speed mode, providing a DDR Clock to the Slave side. The payload data of a burst shall always represent an integer number of

payload data bytes with a minimum length of one byte. Note that for short bursts the Start and End overhead consumes much more time than the actual transfer of the payload data. There is no maximum number of bytes implied by the PHY. However, in the PHY there is no autonomous way of error recovery during a HS data burst and the practical BER will not be zero. Therefore, it is important to consider for every individual protocol what the best choice is for maximum burst length.

1.4.1 Start-of-Transmission

After a Transmit request, a Data Lane leaves the Stop state and prepares for High-Speed mode by means of a Start-of-Transmission (SOT) procedure. Table 3 describes the sequence of events on TX and RX side.

TX Side	RX Side
Drives Stop State (LP-11)	Observes Stop state
Drives HS-Rqst state (LP-01) for time TLPX	Observes transition from LP-11 to LP-01 on the Lines
Drives Bridge state (LP-00) for time THS-PREPARE	Observes transition from LP-01 to LP-00 on the Lines, enables Line Termination after time $TD_{-TERM-EN}$
Enables High-Speed driver and disables Low-Power drivers simultaneously.	
Drives HS-0 for a time THS_{-ZERO}	Enables HS-RX and waits for timer $THS_{-SETTLE}$ to expire in order to neglect transition effects
	Starts looking for Leader-Sequence
Inserts HS Sync-Sequence ‘00011101’ beginning on a rising Clock edge	
	Synchronizes upon recognition of Leader Sequence ‘011101’
Continues to Transmit High-Speed payload data	
	Receives payload data

Table 1.3: Start-of-Transmission Sequence

1.4.2 End-of-Transmission

At the end of a Data Burst, a Data Lane leaves High-Speed Transmission mode and enters the Stop state by means of an End-of-Transmission (EOT) procedure. Table 4 shows a possible sequence of events during the EOT procedure. Note, EOT processing may be handled by the protocol or by the D-PHY.

TX Side	RX Side
Completes Transmission of payload data	Receives payload data
Toggles differential state immediately after last payload data bit and keeps that state for a time $T_{HS\text{-}TRAIL}$	
Disables the HS-TX, enables the LP-TX, and drivesStop state (LP11) for a time $THS_{\text{-}EXIT}$	Detects the Lines leaving LP-00and entering Stop state (LP-11) and disables Termination
	Neglect bits of last period $THS_{\text{-}SKIP}$ to hide transition effects
	Detect last transition in valid Data, determine last valid Data byte and skip trailer sequence

Table 1.4: End-of-Transmission Sequence

1.4.3 HS Data Transmission Burst

Figure 5 shows the sequence of events during the transmission of a Data Burst. Transmission can be started and ended independently for any Lane by the protocol. However, for most applications the Lanes will start synchronously but may end at different times due to an unequal number of transmitted bytes per Lane.

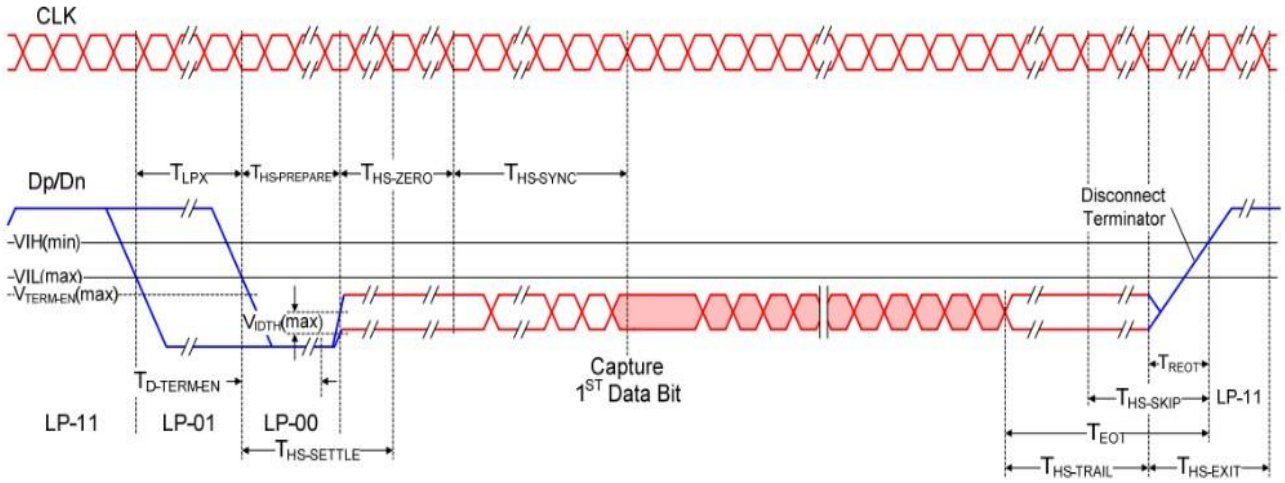


Figure 1.5: HS data transmission

1.4.4 HS Data PPI signals

Symbol	Dir	Categories	Description
High-Speed Transmit Signals			
TxDDRClkHS-I	I	MXXX MCNN	Data Lane High-Speed Transmit DDR Clock. This signal is used to transmit High-Speed data bits over the Lane Interconnect. All Data Lanes use the same TxDDRClkHS-I clock signal.
TxDDRClkHS-Q	I	MCNN	Clock Lane High-Speed Transmit DDR Clock. This signal is used to generate the High-Speed clock signal for the Lane Interconnect. The TxDDRClkHS-Q (quadrature) clock signal is phase shifted from the TxDDRClkHS-I clock signal.
TxRequestHS	I	MXXX SRXX MCNN	HS Request and Data Valid. A low-to-high transition on TxRequestHS triggers the SOT, while a high-to-low transition signals the EOT. On the Clock Lane, TxRequestHS enables HS clock transmission. On Data Lanes, it indicates valid data on TxDataHS, which is accepted when both TxRequestHS and TxReadyHS are high on the same TxByteClkHS rising edge. TxRequestHS stays high until data is accepted and is only active when TxRequestEsc is low.
TxReadyHS	O	MXXX SRXX	High-Speed Transmit Ready. This active high signal indicates that TxDataHS is accepted by the Lane Module to be serially transmitted. TxReadyHS is valid on rising edges of TxByteClkHS.
High-Speed Receive Signals			
RxByteClkHS	O	MRXX SXXX	High-Speed Receive Byte Clock. This is used to synchronize signals in the High-Speed receive clock domain. The RxByteClkHS is generated by dividing the received High Speed DDR clock.

RxDataHS [7:0]	O	MRXX SXXX	High-Speed Receive Data. Eight-bit High-Speed data received by the Lane Module. The signal connected to RxDataHS [0] was received first. Data is transferred on rising edges of RxByteClkHS.
RxActiveHS	O	MRXX SXXX	High-Speed Reception Active. This active high signal indicates that the Lane Module is actively receiving a High-Speed transmission from the Lane interconnect.
RxClkActiveHS	O	MXXX SRX X	Receiver Clock Active. This asynchronous, active high signal indicates that a Clock Lane is receiving a DDR clock signal.
RxValidHS	O	MRXX SXXX	High-Speed Receive Data Valid. This active high signal indicates that the Lane Module is driving data to the protocol on the RxDataHS output. There is no "RxReadyHS" signal, and the protocol is expected to capture RxDataHS on every rising edge of RxByteClkHS where RxValidHS is asserted. There is no provision for the protocol to slow down ("throttle") the receive data.
RxSyncHS	O	MRXX SXXX	Receiver Synchronization Observed. This active high signal indicates that the Lane has seen an appropriate synchronization event. In a typical HS transmission, RxSyncHS is high for one cycle of RxByteClkHS at the beginning of a High-Speed transmission when RxActiveHS is first asserted.
RxDDRClkHS	O	SCNN	Receiver DDR Clock. This is the received DDR clock – it may be used by the protocol if required. This signal is low whenever RxClkActiveHS is low.

Table 1.5: HS mode signals

1.4.5 High-Speed Transmit from the Master Side

Figure 6 shows an example of a High-Speed transmission on the Master side. During High-Speed transmission, TxRequestHS must be high for the Lane Module to accept TxDataHS. Transmission starts when the protocol sets TxDataHS and asserts TxRequestHS. Each data byte is accepted on a rising TxByteClkHS edge when TxReadyHS is high. The protocol continues providing data or ends transmission by deasserting TxRequestHS. Transmission can consist of as few as one byte.

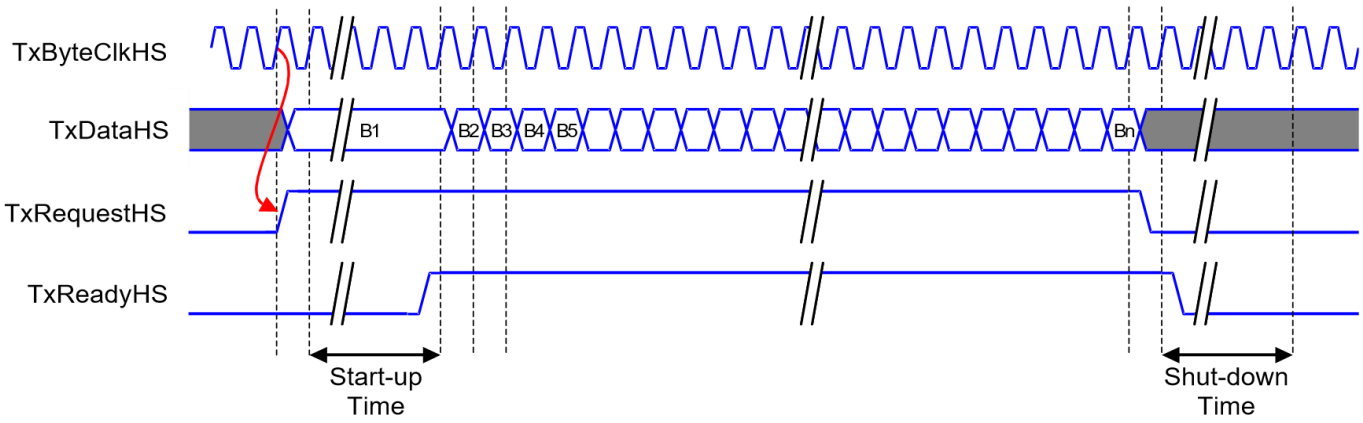


Figure 1.6: Example High-Speed Transmission from the Master Side

1.4.6 HS Data Reception Example

Figure 7 shows an example of a High-Speed reception at the Slave side. The RxActiveHS signal indicates that a receive operation is occurring. A normal reception starts with a pulse on RxSyncHS followed by valid receive data on subsequent cycles of RxByteClkHS. Note that the protocol is prepared to receive all of the data. There is no method for the receiving protocol to pause or slow data reception.

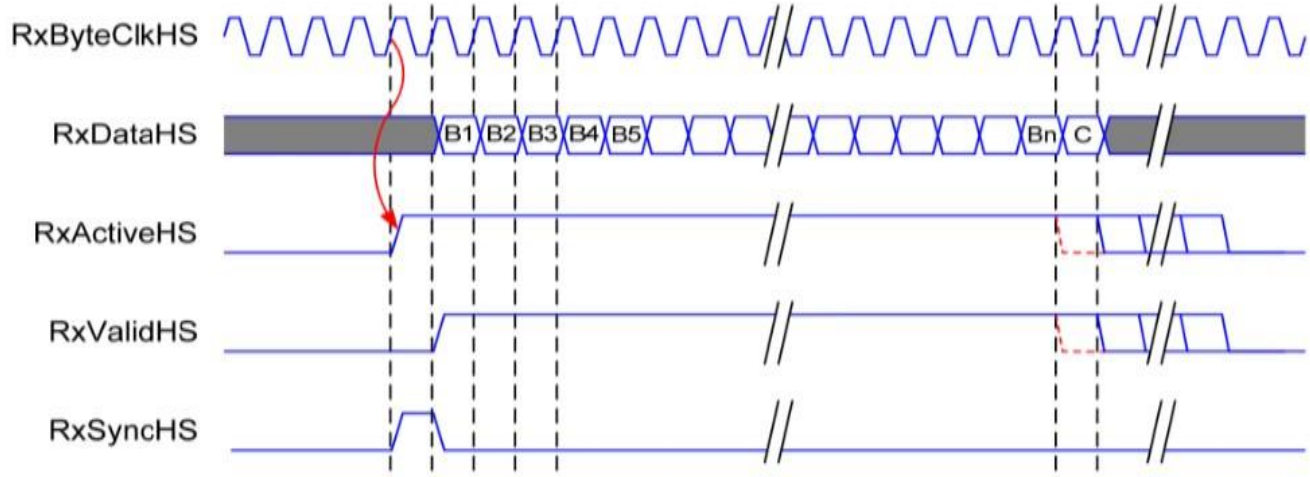


Figure 1.7: Example High-Speed Transmission

1.5 ESCAPE MODE DATA LANE

Escape mode is a special mode of operation for Data Lanes using Low Power states. Escape mode operation shall be supported in the Forward direction and is optional in the Reverse direction. If supported, Escape mode does not have to include all available features. A Data Lane shall enter Escape mode via an Escape mode Entry procedure (LP-11, LP-10, LP-00, LP-01, LP-00). As soon as the final Bridge state (LP-00) is observed on the Lines the Lane shall enter Escape mode in Space state (LP-00). If an LP-11 is detected at any time before the final Bridge state (LP-00), the Escape mode Entry procedure shall be aborted and the receive side shall wait for, or return to, the Stop state. For Data Lanes, once Escape mode is entered, the transmitter shall send an 8-bit entry command to indicate the requested action. Table 6 lists all currently available Escape mode commands and actions. All unassigned commands are reserved for future expansion. The Stop state shall be used to exit Escape mode and cannot occur during Escape mode operation because of the Spaced-One-Hot encoding. The stop state immediately returns the Lane to Control mode. If the entry command doesn't match the supported command, that Escape mode action shall be ignored and the receive side waits until the transmit side returns to the Stop state. The PHY in Escape mode shall apply Spaced-One-Hot bit encoding for asynchronous communication. Therefore, the operation of a Data Lane in this mode does not depend on the Clock Lane. The complete Escape mode action for a Trigger-Reset command is shown in Figure. Spaced-One-Hot coding means that each Mark state is interleaved with a Space state. Each symbol consists therefore of two parts: a One-Hot phase (Mark-0 or Mark-1) and a Space phase. The TX shall send Mark-0 followed by a Space to transmit a 'zero-bit' and it shall send a Mark-1 followed by a Space to transmit a 'one-bit'. A Mark that is not followed by a Space does not represent 32 a bit. The last phase before exiting Escape mode with a Stop state shall be a Mark-1 state that is not part of the communicated bits, as it is not followed by a Space state. The Clock can be derived from the two-Line signals, Dp and Dn, by means of an XOR function. The length of each individual LP state period shall be at least $T_{LPX, MIN}$.

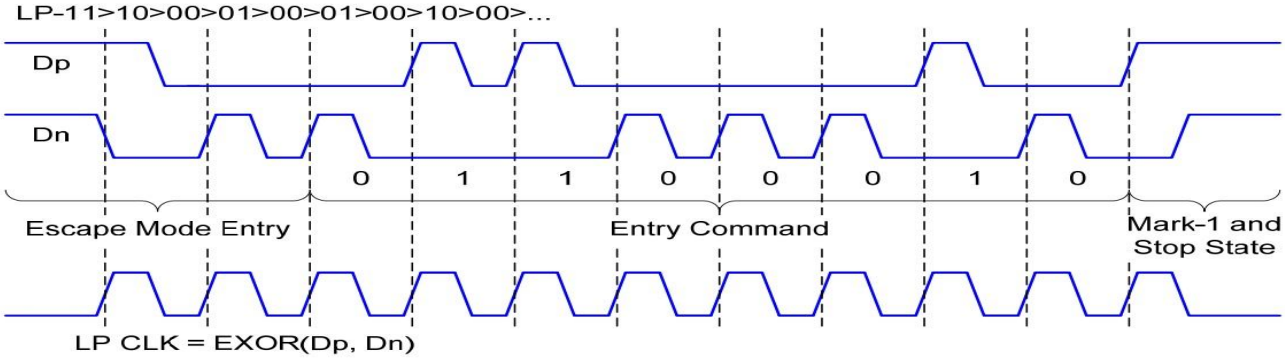


Figure 1.8: Trigger-Reset Command in Escape Mode

Escape Mode Action	Command Type	Entry Command Pattern
Low-Power Data Transmission	Mode	11100001
Ultra-Low Power State	Mode	00011110
Undefined-1	Mode	10011111
Undefined-2	Mode	11011110
Reset-Trigger [Remote Application]	Trigger	01100010
Unknown-3	Trigger	01011101
Unknown-4	Trigger	00100001
Unknown-6	Trigger	10100000

Table 1.6: Escape Entry Codes

1.5.1 Remote Triggers

Trigger signaling is the mechanism to send a flag to the protocol at the receiving side, on request of the protocol on the transmitting side. This can be either in the Forward or Reverse direction depending on the direction of operation and available Escape mode functionality. Trigger signaling requires Escape mode capability and at least one matching Trigger Escape Entry Command on both sides of the interface. Figure shows an example of an Escape mode Reset-Trigger action. The Lane enters Escape mode via the Escape mode Entry procedure. If the Entry Command Pattern matches the Reset-Trigger Command a Trigger is flagged to the protocol at the receive side via the logical PPI. Any bit received after a Trigger Command but before the Lines go to stop state shall be ignored. Therefore, dummy bytes can be concatenated to provide Clock information to the receive side. Note that Trigger signaling including Reset-Trigger, is a generic messaging system. The Trigger commands do not impact the behavior of the PHY itself. Therefore, Triggers can be used for any purpose by the Protocol layer.

1.5.2 Low-Power Data Transmission

For Low-Power data transmission the TxClkEsc is used instead of TxDDRClkHS-I/Q and TxByteClkHS. Furthermore, while the High-Speed interface signal TxRequestHS serves as both a transmit request and a data valid signal, on the Low-Power interface two separate signals are used. The Protocol directs the Data Lane to enter Low-Power data transmission Escape mode by asserting TxRequestEsc with TxLpdtEsc high. The Low-Power transmit data is transferred on the TxDataEsc lines when TxValidEsc and TxReadyEsc are both active at a rising edge of TxClkEsc. The byte is transmitted in the time after the TxDataEsc is accepted by the Lane Module (TxValidEsc = TxReadyEsc = high) and therefore the TxClkEsc continues running for some minimum time after the last byte is transmitted. The Protocol knows the byte transmission is finished when TxReadyEsc is asserted. After the last byte has been transmitted, the protocol de-asserts

TxRequestEsc to end the Low-Power data transmission. This causes TxReadyEsc to return low, after which the TxClkEsc clock is no longer needed. Whenever TxRequestEsc transitions from high-to-low, it always remains in the low state for a minimum of two TxClkEsc clock cycles. Figure 9 shows an example of Low-Power data transmission operation.

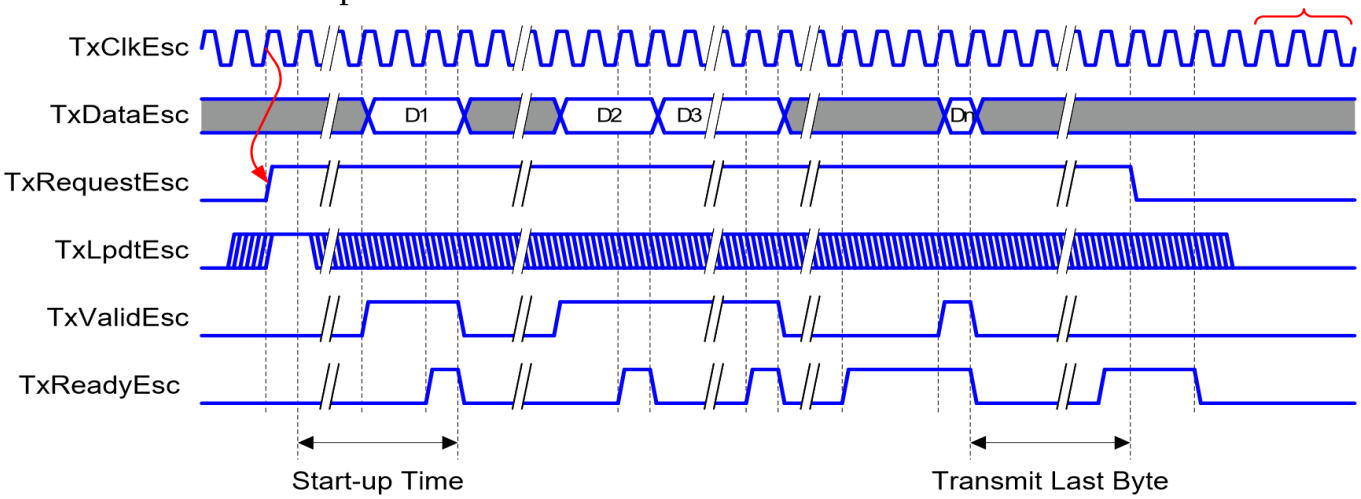


Figure 1.9: Low-Power Data Transmission

1.5.3 Low-Power Data Reception

Figure 10 shows an example of Low-Power data reception. In this example, a Low-Power escape "clock" is generated from the Lane Interconnect by the logical exclusive-OR of the Dp and Dn lines. This "clock" is used within the Lane Module to capture the transmitted data. In this example, the "clock" is also used to generate RxClkEsc. The signal RxLpdtEsc is asserted when the escape entry command is detected and stays high until the Lane returns to Stop state, indicating that the transmission has finished. It is important to note that because of the asynchronous nature of Escape mode transmission, the RxClkEsc signal can stop at any time in either the high or low state. This is most likely to happen just after a byte has been received, but it could happen at other times as well.

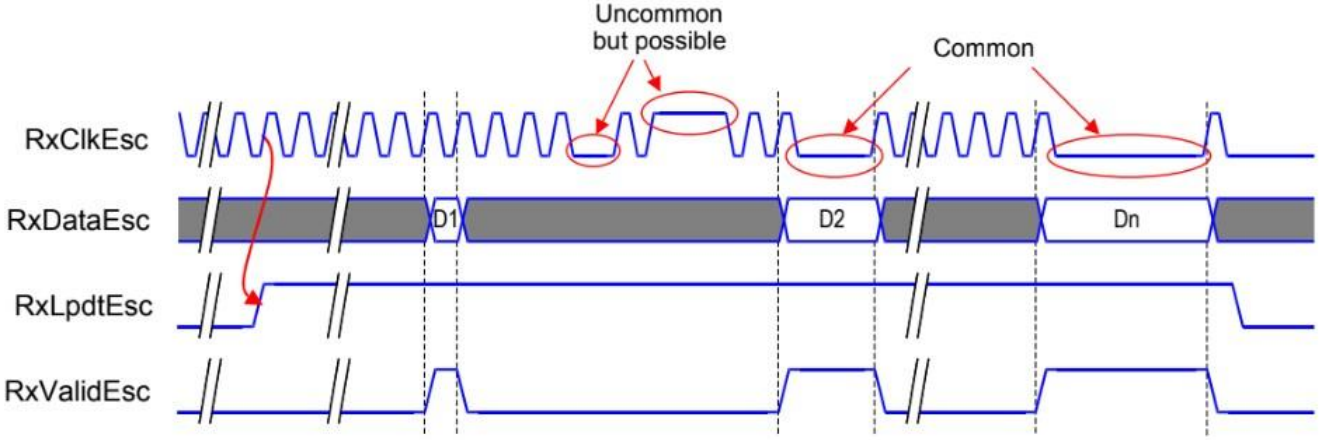


Figure 1.10: Two Data Byte Low Power Data Reception

Using LPDT, a Low-Power (Bit) Clock signal ($f_{MOMENTARY}$ < 20MHz) provided to the transmit side is used to transmit data. Data reception is self-timed by the bit encoding; therefore, a variable clock rate is allowed. At the end of LPDT, the Lane shall return to the Stop state.

1.5.4 Ultra-Low Power State

If the Ultra-Low Power State Entry Command is sent after an Escape mode Entry command, the Lane shall enter the Ultra-Low Power State (ULPS). This command shall be flagged to the receive side Protocol.
During this state, the Lines are in the Space state (LP-00). Ultra-Low Power State is excited by means of a Mark-1 state with a length T_{WAKEUP} followed by a Stop state.

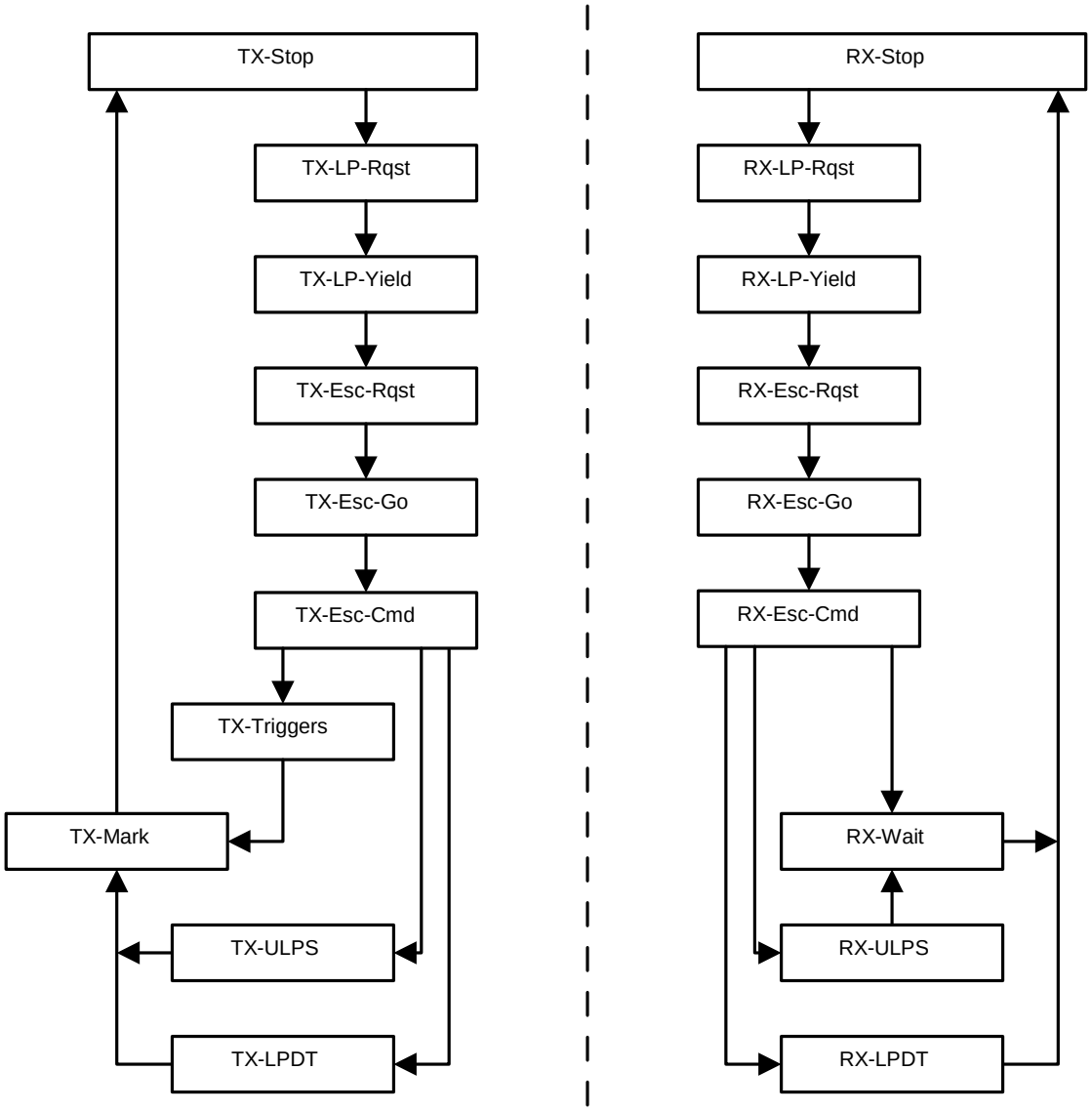


Figure 1.11: Escape mode state machine

State	Line Condition/State	Exit State	Exit Conditions
Any RX state	Any Received	RX-Stop	Observe LP-11 at Lines
RX-Stop	Receive LP-11	RX-LP-Rqst	Line transition to LP-10
RX-LP-Rqst	Receive LP-10	RX-LP-Yield	Line transition to LP-00
RX-LP-Yield	Receive LP-00	RX-Esc-Rqst	Line transition to LP-01
RX-Esc-Rqst	Receive LP-01	RX-Esc-Go	Line transition to LP-00
RX-Esc-Go	Receive LP-00	RX-Esc-Cmd	Line transition out of LP-00
RX-Esc-Cmd	Receive sequence of 8-bit (16-line states) One Spaced Hot Encoded Entry Command	RX-Wait	After Trigger and Unrecognized Commands
		RX-ULPS	After Ultra-Low Power Command
		RX-LPDT	After LP Data Transmission Command
RX-ULPS	Receive LP-00	RX-Wait	Line transition to LP-10
RX-LPDT	Receive serial, One Spaced Hot Encoded payload data	RX-Stop	Line transition to LP-11 (Last state should be a Mark-1)
RX-Wait	Any, except LP11	RX-Stop	Line transition to LP-11

Table 1.7: Escape mode state machine

1.6 Bi-directional Data Lane Turnaround

The transmission direction of a bi-directional Data Lane can be swapped by means of a Link Turnaround procedure. This procedure enables information transfer in the opposite direction of the current direction. The procedure is the same for either a change from Forward-to-Reverse direction or Reverse-to-Forward direction. Notice that Master and Slave side shall not be changed by Turnaround. Link Turnaround handled completely in Control mode.

Initial TX Side = Final RX Side	Initial RX Side = Final TX Side
Drives Stop State (LP-11)	Observes Stop state
Drives LP-Rqst state (LP-10) for T_{LPX}	Observes transition from LP-11 to LP-10 states
Drives Bridge state (LP-00) for T_{LPX}	Observes transition from LP-10 to LP-00 states
Drives LP-10 for a time T_{LPX}	Observes transition from LP-00 to LP-10 states
Drives Bridge State (LP-00) for a time T_{TA-GO}	Observes the transition from LP-10 to Bridge State and waits for a time $T_{TA-SURE}$.
	Drives Bridge state (LP-00) for a period T_{TAGET}

Stop driving the Lines and observe the Line states with its LP-RX to see an acknowledgement.	
	Drives LP-10 for a period T_{LPX}
Observes LP-10 on the Lines, interprets this as an acknowledge that the other side has indeed taken control. Waits for Stop State to complete the Turnaround procedure.	
	Drives Stop state (LP-11) for a period T_{LPX}
Observes transition to Stop state (LP-11) on the Lines, interprets this as Turnaround completion acknowledgement, switches to normal LP receive mode and waits for further actions from the other side	

Table 1.8: Link Turnaround Sequence

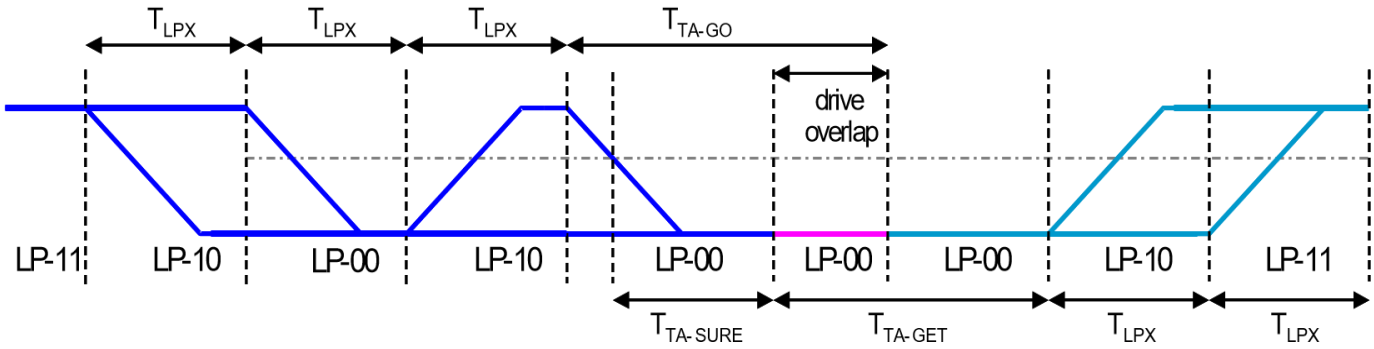


Figure 1.12: Turnaround Procedure

The Low-Power clock timing for both sides of the Link does not have to be the same but may differ. However, the ratio between the Low-Power State Periods, TLPX, is constrained to ensure proper Turnaround behavior. The Turnaround procedure can be interrupted if the Lane is not yet driven into TX-LP-Yield by means of driving a Stop state. Driving the Stop State shall abort the Turnaround procedure and return the Lane to the Stop State. The PHY shall ensure against interruption of the procedure after the end of TX-TA-Rqst, RX-TA-Rqst, or TX-TA-GO. Once the PHY drives TX-LP-Yield, it shall not abort the Turnaround procedure. The Protocol may take appropriate action if it determines an error has occurred because the Turnaround procedure did not complete within a certain time.

Figure 13 shows the Turnaround state machine that is described in Table 8.

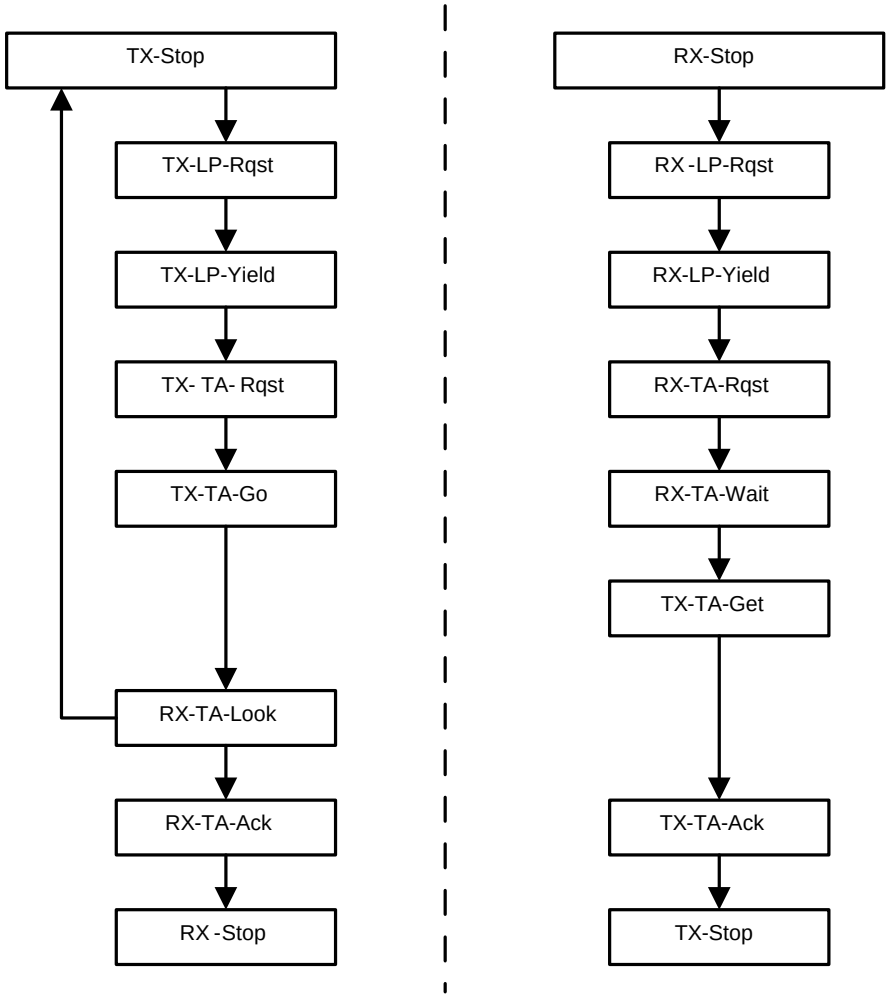


Figure 1.13: Turnaround State Machine

If the Master side and Slave side Lane Modules are both bi-directional, it is possible to turn around the Link for High-Speed and/or Escape mode signaling. As explained, which side is allowed to transmit is determined by passing a “token” back and forth. That is, the side currently transmitting passes the token to the receiving side. If the receiving side acknowledges the turn-around request, as indicated by driving the appropriate line state, the direction is switched. Figure 14 shows an example of two turn-around events. At the beginning, the local side is the transmitter, as shown by Direction=0. When the protocol on this side wishes to turn the Lane around (i.e. give the token to the other side), it asserts TurnRequest for at least one cycle of TxClkEsc. This initiates the turn-around procedure. The remote side acknowledges the turn-around request by driving the appropriate states on the Lines. When this happens, the local Direction signal changes from transmit (0) to receive (1). Later in the example of Figure 14, the remote side initiates a turn-around request, passing the token back to the local side. When this happens, the local Direction signal changes back to transmit (0). Note that there is no prescribed way for a receiver to

request access to the Link. The current transmitter is in control of the Link direction and decides when to turn the Link around, passing control to the receiver. If the remote side does not acknowledge the turn-around request, the Direction signal does not change.

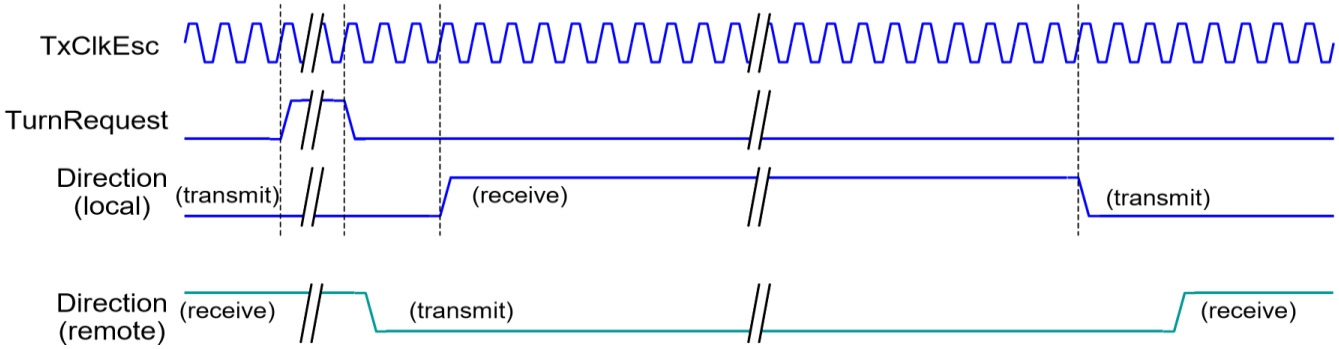


Figure 1.14: Example Turn-around Actions Transmit-to-Receive and Back to Transmit

1.7 Escape Mode PPI signals

Symbol	Dir	Categories	Description
Escape Mode Transmit Signals			
TxClkEsc	I	MXXX SXXY	Escape mode Transmit Clock. This clock is used to generate escape sequences. Its period is constrained by the D-PHY specification. It also synchronizes TurnRequest and is required in modules with bi-directional High-Speed support, even if they don't support escape mode.
TxRequestEsc	I	MXXX SXXY	Escape mode Transmit Request. This asserted together with exactly one of TxLpdtEsc, TxUlpsEsc, or one bit of TxTriggerEsc, is used to request entry into ESC mode. Lane stays in escape mode until TxRequestEsc is de-asserted. TxRequestEsc is only asserted by the protocol while TxRequestHS is low.
TxLpdtEsc	I	MXAX SXXA	Escape mode Transmit Low-Power Data. This active high signal is asserted with TxRequestEsc to cause the Lane Module to enter Low-Power data transmission mode. The Lane Module remains in this mode until TxRequestEsc is deasserted. TxUlpsEsc and all bits of TxTriggerEsc are low when TxLpdtEsc is asserted.

TxUlpsExit	I	MXXX SXXY MCNN	Transmit ULP Exit Sequence. This active high signal is asserted when ULPS is active, and the protocol is ready to leave ULPS. The PHY leaves ULP state and begins driving Mark-1 after TxUlpsExit is asserted. The PHY later drives the Stop state (LP-11) when TxRequestEsc is deasserted. TxUlpsExit is synchronous to TxClkEsc. This signal is ignored when the Lane is not in the ULP State.
TxUlpsEsc	I	MXXX SXXY	Escape mode Transmit Ultra-Low Power State. This active high signal is asserted with TxRequestEsc to cause the Lane Module to enter the Ultra-Low Power State. The Lane Module remains in this mode until TxRequestEsc is de-asserted. TxLpdtEsc and all bits of TxTriggerEsc are low when TxUlpsEsc is asserted.
TxTriggerEsc [3:0]	I	MXXX SXXY	Escape mode Transmit Trigger 0-3. One of these active-high signals is asserted with TxRequestEsc to send the associated Trigger across the Lane. In the receiving Lane Module, the same RxTriggerEsc bit is asserted and remains so until the interconnect returns to Stop state when TxRequestEsc is de-asserted. Only one bit of TxTriggerEsc is asserted at a time, and only when TxLpdtEsc and TxUlpsEsc are low.
TxDataEsc [7:0]	I	MXAX SXXA	Escape mode Transmit Data. This is the eight-bit escape mode data to be transmitted in Low-Power data transmission mode. The signal connected to TxDataEsc [0] is transmitted first. Data is captured on the rising edges of TxClkEsc.
TxValidEsc	I	MXAX SXXA	Escape mode Transmit Data Valid. This signal indicates that the protocol is driving valid data on TxDataEsc to be transmitted. The Lane Module accepts the data when TxRequestEsc, TxValidEsc and TxReadyEsc are all active on the same rising TxClkEsc clock edge.

TxValidEsc	I	MXAX SXXA	Escape mode Transmit Data Valid. This active high signal indicates that the protocol is driving valid data on TxDataEsc to be transmitted. The Lane Module accepts the data when TxRequestEsc, TxValidEsc and TxReadyEsc are all active on the same rising TxClkEsc clock edge.
TxReadyEsc	O	MXAX SXXA	Escape mode Transmit Ready. This active high signal indicates that TxDataEsc is accepted by the Lane Module to be serially transmitted. TxReadyEsc is valid on rising edges of TxClkEsc.
Escape Mode Receive Signals			
RxClkEsc	O	MXXY SXXX	Escape mode Receive Clock. This signal is used to transfer received data to the protocol during escape mode. This "clock" is generated from the two Low-Power signals in the Lane interconnect. Because of the asynchronous nature of Escape mode data transmission, this "clock" may not be periodic.
RxUlpsEsc	O	MXXY SXXX	Escape Ultra-Low Power (Receive)mode. This active high signal is asserted to indicate that the Lane Module has entered the ULPS.
RxLpdtEsc	O	MXXA SXAX	Escape Low-Power Data Receive mode. This active-high signal indicates the Lane Module is in Low-Power data receive mode. Data appears on RxDataEsc when RxValidEsc is active. The Lane stays in this mode while RxLpdtEsc is high and exits upon detecting a Stop state.
RxTriggerEsc [3:0]	O	MXXY SXXX	Escape mode Receive Trigger 0-3. These active high signals indicate that a trigger event has been received. The asserted RxTriggerEsc signal remains active until a Stop state is detected on the Lane interconnect.
RxDataEsc [7:0]	O	MXXA SXAX	Escape mode Receive Data. This is the eight-bit escape mode Low-Power data received by the Lane Module. Data is transferred on the rising edges of RxClkEsc.

RxValidEsc [3:0]	O	MXXA SXAX	Escape mode Receive Data Valid. is an active high signal indicating the Lane Module sends valid data via RxDataEsc. With no RxReadyEsc signal, the protocol must capture RxDataEsc on rising RxClkEsc edges when RxValidEsc is asserted and cannot throttle the data.

Control Signals			
TurnRequest	I	XRXX XFXY	Turn Around Request. This active high signal is used to indicate that the protocol desires to turn the Lane around, allowing the other side to begin transmission. TurnRequest is valid on rising edges of TxClkEsc. TurnRequest is only meaningful for a Lane Module that is currently the transmitter (Direction=0). If the Lane Module is in receive mode (Direction=1), this signal is ignored.
Direction	O	XRXX XFXY	Transmit/Receive Direction. This signal is used to indicate the current direction of the Lane interconnect. When Direction=0, the Lane is in transmit mode (0=Output). When Direction=1, the Lane is in receive mode (1=Input).
TurnDisable	I	XRXX XFXY	Disable Turn-around. This signal is used to prevent a (bi-directional) Lane from going into transmit mode – even if it observes a turn-around request on the Lane interconnect. This is useful to prevent a potential “lockup” situation when a unidirectional Lane Module is connected to a bi-directional Lane Module.
Enable	I	XXXX XCNN	Enable is an active-high, level-sensitive signal that takes the Lane Module out of shutdown. When low, all drivers, receivers, and detectors are off, PPI inputs are ignored, and outputs are set to inactive. It operates independently of any clock.

ForceRxmode	I	MRXX MXXY SXXX	Force Lane Module into Receive mode / Wait for Stop State. This signal allows the protocol to initialize a Lane, or force a bi-directional Lane, into receive mode. This signal is used during initialization or to resolve a contention situation. When this signal is high, the Lane Module immediately transitions into receive control mode and waits for a Stop state to appear on the Lane. When used for initialization, this signal should be released, i.e. driven low, only when the Dp & Dn inputs are in Stop state for a time T_{INIT}, or longer.
ForceTx Stopmode	I	MXXX SRXX SXXY	Force Lane Module into Transmit mode / Generate Stop state. This signal allows the protocol to force a Lane Module into transmit mode and Stop state during initialization or following an error situation, e.g. expired time out. When this signal is high, the Lane Module immediately transitions into transmit mode and the module state is forced into the Stop state.
Stopstate	O	XXXX XCNN	Lane is in Stop state. This active high signal indicates that the Lane Module, regardless of whether the Lane Module is a transmitter or a receiver, is currently in Stop state. Note that this signal is asynchronous to any clock in the PPI interface. Also, the protocol may use this signal to indirectly determine if the PHY line levels are in the LP-11 state.
TxUlpsClk	I	MCNN	Transmit Ultra-Low Power State on Clock Lane. This active high signal is asserted to cause a Clock Lane Module to enter the Ultra-Low Power State. The Lane Module remains in this mode until TxUlpsClk is de-asserted.

UlpsActiveNot	O	XXXX XCNN	ULP State (not) Active. This active low signal is asserted to indicate that the Lane is in ULP state. For a transmitter, this signal is asserted sometime after TxUlpsEsc and TxRequestEsc (TxUlpsClk for a Clock Lane) are asserted. The transmitting PHY continues to supply TxClkEsc until UlpsActiveNot is asserted. In order to leave ULP state, the transmitter first drives TxUlpsExit high, then waits for UlpsActiveNot to become high (inactive). At that point, transmitting PHY is active and has started transmitting a Mark-1 on the Lines. The protocol waits for a time Twakeup and then drives TxRequestEsc (TxUlpsClk) inactive to return the Lane to Stop state. For a receiver, this signal indicates that the Lane is in ULP state. At the beginning of ULP state, UlpsActiveNot is asserted together with RxUlpsEsc, or RxUlpsClkNot for a Clock Lane. At the end of the ULP state, this signal becomes inactive to indicate that the Mark-1 state has been observed. Later, after a period Twakeup, the RxUlpsEsc (or RxUlpsClkNot) signal is deasserted.
RxUlpsClkNot	O	SCNN	Receive Ultra-Low Power State on Clock Lane. This active low signal is asserted to indicate that the Clock Lane Module has entered the Ultra-Low Power State. The Lane Module remains in this mode with RxUlpsClkNot asserted until a Stop state is detected on the Lane Interconnect.
Error Signals			
ErrSotHS	O	MRXX SXXX	Start-of-Transmission (SoT) Error. If the High-Speed SoT leader sequence is corrupted, but in such a way that proper synchronization can still be achieved, this active high signal is asserted for one cycle of RxByteClkHS. This is considered to be a “soft error” in the leader sequence and confidence in the payload data is reduced.

ErrSotSyncHS	O	MRXX SXXX	Start-of-Transmission Synchronization Error. If the High-Speed SoT leader sequence is corrupted in a way that proper synchronization cannot be expected, this active high signal is asserted for one cycle of RxByteClkHS.
ErrEsc	O	MXXY SXXX	Escape Entry Error. If an unrecognized escape entry command is received, this active high signal is asserted and remains asserted until the next change in line state.
ErrSyncEsc	O	MXXA SXAX	Low-Power Data Transmission Synchronization Error. If the number of bits received during a Low-Power data transmission is not a multiple of eight when the transmission ends, this active high signal is asserted and remains asserted until the next change in line state.
ErrControl	O	MXXY SXXX	Control Error. This active high signal is asserted when an incorrect line state sequence is detected. For example, if a turn-around request or escape mode request is immediately followed by a Stop state instead of the required Bridge state, this signal is asserted and remains asserted until the next change in line state.
ErrContentLP0	O	MXXX SXXY	LP0 Contention Error. This active high signal is asserted when the Lane Module detects a contention situation on a line while trying to drive the line low.
ErrContentLP1	O	MXXX SXXY	LP1 Contention Error. This active high signal is asserted when the Lane Module detects a contention situation on a line while trying to drive the line high.

Table 1.9: PPI signals

CHAPTER 2: DESIGN

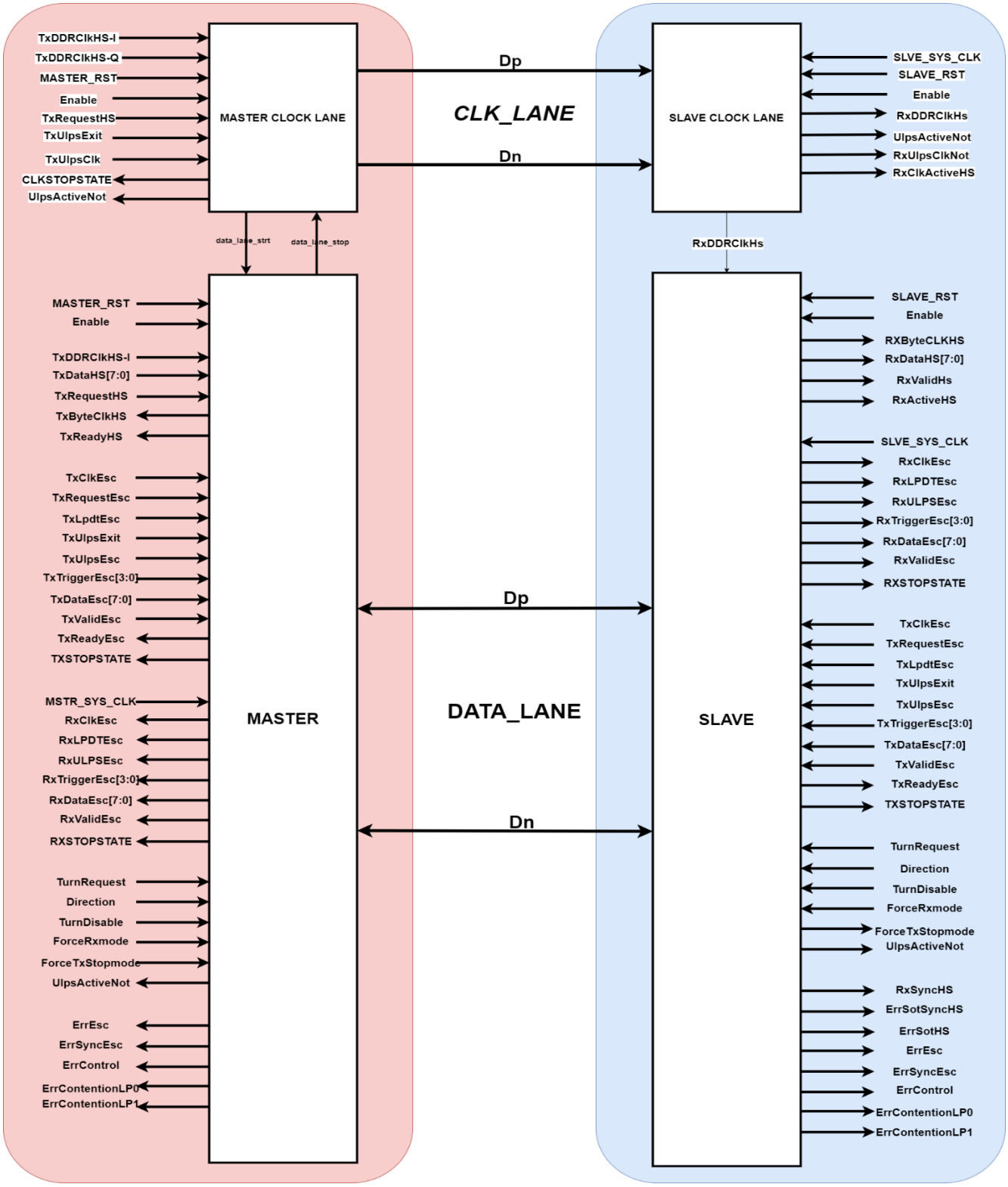


Figure 2.1: MIPI D PHY TOP SYSTEM

2.1 MIPI D-PHY CLOCK LANE TRANSMITER TOP LEVEL

The Clock Lane in the MIPI D-PHY transmitter is a critical component responsible for delivering a continuous, high-speed differential clock signal (CLK_Dp, CLK_Dn) that ensures synchronization across all active data lanes during high-speed transmission. the Clock Lane is unidirectional and exists only on the master side of the link. It supports both High-Speed (HS) transmission and Ultra-Low Power State (ULPS) for power-efficient idle periods.

The proposed design of the Clock Lane Transmitter is composed of three main architectural blocks: a finite state machine (FSM), a timing control counter, and a multiplexer-based output driver. Together, these components enable precise control over mode transitions, timing compliance, and reliable clock generation.

2.1.1. Block Diagram

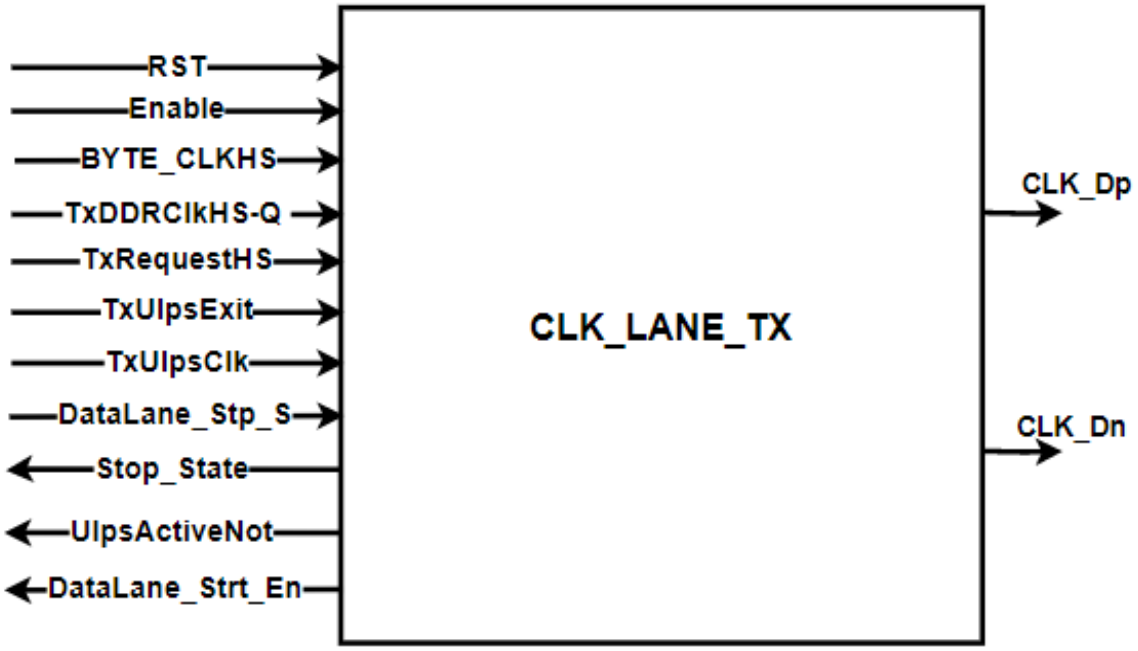


Figure 2.2: CLK Lane TX block

2.1.2. Interface Table

Port	Direction	Width	Description
Byte_clk_hs	Input	1-bit	Byte clock
rst	Input	1-bit	System reset
Enable	Input	1-bit	clock lane Enable
TxRequestHS	Input	1-bit	High speed clock request
TxUlpsClk	Input	1-bit	Ultra-low power state enter request
TxUlpsExit	Input	1-bit	Ultra-low power state exit request
TxDDRClkHS	Input	1-bit	Double data rate clock (200M)
Data_lane_stop	Input	1-bit	Indicate data lane has successfully completed HS data transmission.
Stop_state	Output	1-bit	Stop state
UlpsActiveNot	Output	1-bit	Active low signal. ULPS active
Data_lane_strt_En	Output	1-bit	Indicate The data lane can start high-speed data transmission
CLK_DP, CLK_DN	Output	1-bit	Output Of Master CLK Lane (Transmitter DDR CLK_Q).

Table 2.1: Interface table of CLK lane TX

2.1.3 Architecture

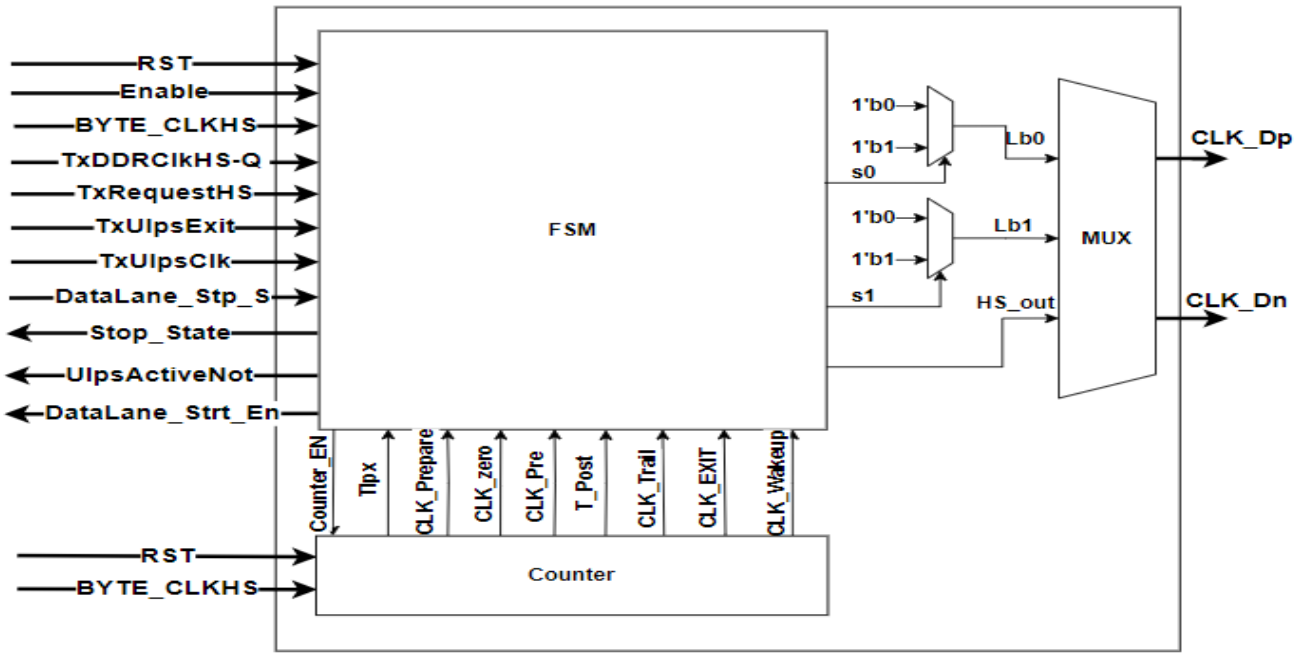


Figure 2.3: Components of CLK Lane TX block

Finite State Machine (FSM)

At the heart of the Clock Lane transmitter is a finite state machine that governs all operational states, including initialization, entry and exit from ULPS, high-speed preparation, and active clock transmission. The FSM receives several control signals from the protocol layer and other PHY modules, including:

- Enable and RST for general control,
- TxRequestHS to initiate high-speed transmission.
- TxUlpsClk and TxUlpsExit for transitioning in and out of ultra-low power mode
- BYTE_CLKHS and TxDDRClkHS_Q as reference clocks for state timing and DDR clock generation.

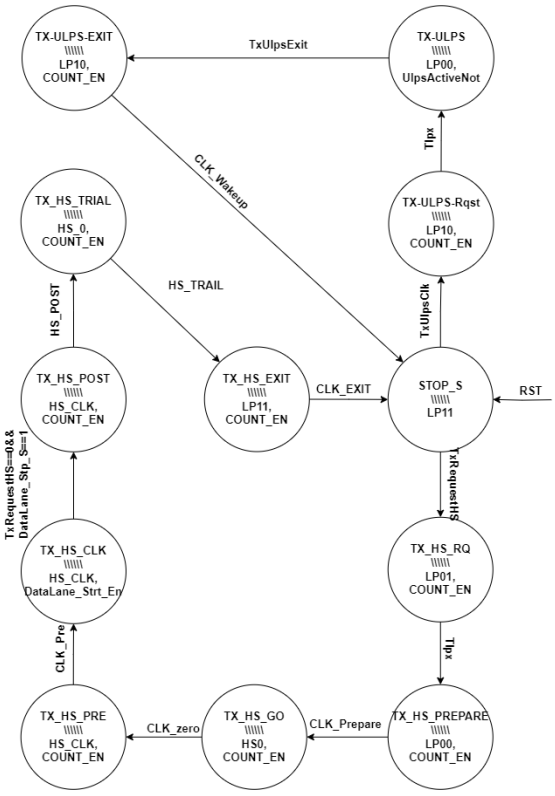


Figure 2.4: CLK Lane FSM

FSM State Descriptions:

- STOP_S: Idle state where the clock lane drives LP-11.
- TX_HS_RQ: Begins high-speed entry by driving LP-01 and enabling the counter.
- TX_HS_PREPARE: Drives LP-00 during the prepare phase for HS transmission.
- TX_HS_GO: Starts high-speed clock toggling by outputting HS0.
- TX_HS_PRE: Generates the preamble with HS_CLK before active clocking.
- TX_HS_CLK: Transmits the continuous HS clock while data lanes are active.
- TX_HS_POST: Continues HS clock after transmission to satisfy T_POST timing.
- TX_HS_TRIAL: Final HS clock pulses before transitioning back to LP mode.
- TX_HS_EXIT: Drives LP-11 to exit high-speed transmission.
- TX_ULPS_Rqst: ULPS begins entry by driving LP-10 and enabling the counter.
- TX_ULPS: Enters ULPS mode with static LP-00 on the clock lane.
- TX_ULPS_EXIT: Exits ULPS by driving LP-10 and initiating CLK_Wakeup.

Output Driver Logic (MUX)

The FSM controls a pair of multiplexers to manage the Clock Lane outputs based on the operating mode. In HS mode, it routes the DDR clock (TxDDRClkHS_Q) to generate the differential clock signal. In LP and ULPS modes, as well as during HS entry sequences, the FSM drives static values corresponding to LP-0 and LP-1 through the multiplexers.

Timing Control (Counter Module)

A dedicated counter module is integrated to manage the precise timing requirements of the various protocol-defined phases, including:

Parameter	Description	Time
TLPX	Transmitted length of any Low-Power state period	(50ns MIN) 60ns
TCLK_PREPARE	Time that the transmitter drives the Clock Lane LP-00 Line State immediately before the HS-0 Line State starts the HS transmission.	38 → 95 ns (80ns)
TCLK_ZERO	TCLK_PREPARE + Time that the transmitter drives the HS-0 state prior to starting the Clock.	300ns (MIN) TCLK_ZERO = 220ns
TCLK-PRE	The transmitter must drive the HS clock briefly before Data Lanes switch from LP to HS mode.	8 UI (UI = 2.5ns) (1 cycle)
TCLK-POST	Time that the transmitter continues to send HS clock after the last associated Data Lane has transitioned to LP Mode.	60ns + 52*UI (190ns MIN), 200ns
TCLK-TRAIL	Time that the transmitter drives the HS-0 after the last payload clock bit of a HS transmission burst.	60ns
THS-EXIT	Time that the transmit LP-11 following a HS burst.	100ns
TWAKEUP	Time that a transmit a Mark-1 prior to a Stop state to initiate an exit from ULPS.	1ms

Table 2.2: Timing Parameters of Counter of CLK lane TX

The FSM enables the counter during each transition phase to ensure all signal timing parameters strictly adhere to the global timing constraints specified in Section 6.9 and Table 14 of the MIPI D-PHY specification. This guarantees interoperability and signal integrity across the link.

2.2 MIPI D-PHY CLOCK LANE RECEIVER TOP LEVEL

The main purpose of the CLK Lane Receiver in MIPI D-PHY is to receive the differential high-speed clock transmitted from the TX side. It provides a precise timing reference used to sample and deserialize data from the data lanes. This ensures synchronization and alignment during high-speed (HS) DDR data transmission. It also detects mode transitions (HS, ULPS) to manage power and control the receiver state machine. Without it, the system cannot guarantee correct timing for high-speed data reception. The proposed design of the Clock Lane Receiver is composed of three main architectural blocks: a finite state machine (FSM), a timing control counter, and a Comparator Together, these components enable precise control over mode transitions, timing compliance, and received reliable clock.

2.2.1. Block Diagram

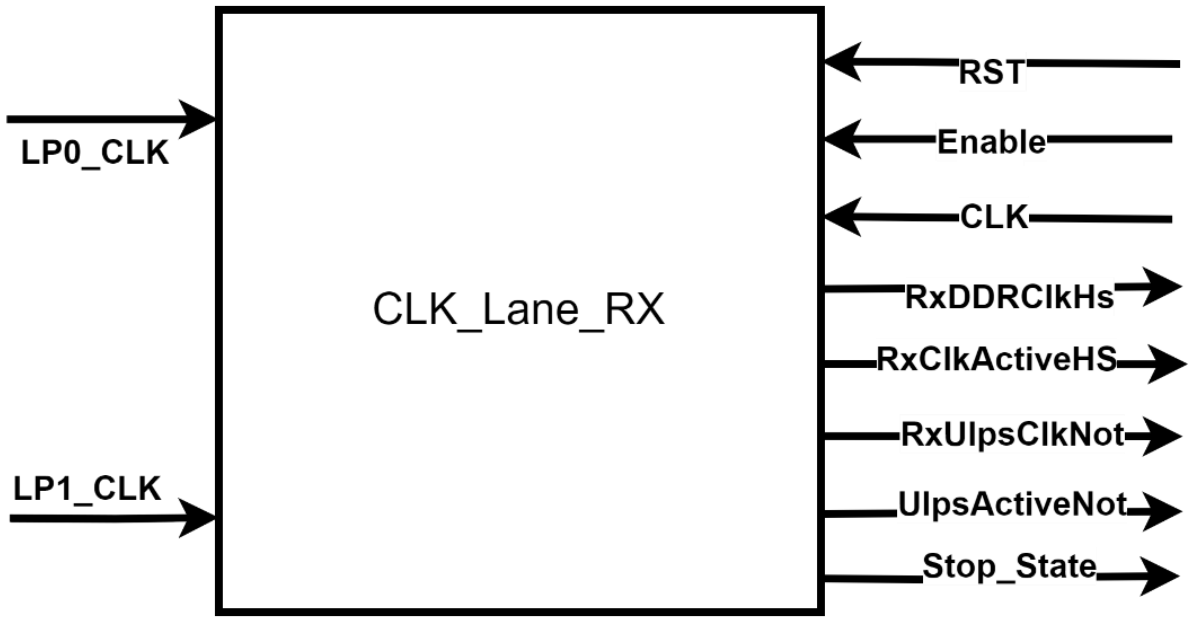


Figure 2.5: CLK Lane RX block

2.2.2. Interface Table

Port	Direction	Width	Description
RST	Input	1-bit	System reset
Enable	Input	1-bit	clock lane Enable
CLK	Input	1-bit	The main Clock of clock lane
CLK_DP, CLK_DN	Output	1-bit	Receive DDR CLK Q from master CLK lane
RxDDRClkHS	Output	1-bit	This is the received DDR clock

RxClkActiveHS	Output	1-bit	This asynchronous, active high signal indicates that a Clock Lane is receiving a DDR clock signal.
RxUlpsClkNot	Output	1-bit	This active low signal is asserted to indicate that the Clock Lane has entered the (ULPS). The Lane remains in this mode with RxUlpsClkNot asserted until a Stop state is detected.
UlpsActiveNot	Output	1-bit	This active low signal indicates the Lane is in ULPS. It is asserted together with RxUlpsClkNot. This signal becomes inactive to indicate Mark-1 state has been observed.
Stop state	Output	1-bit	Stop state

Table 2.3: Interface table of CLK lane RX

2.2.3 Architecture

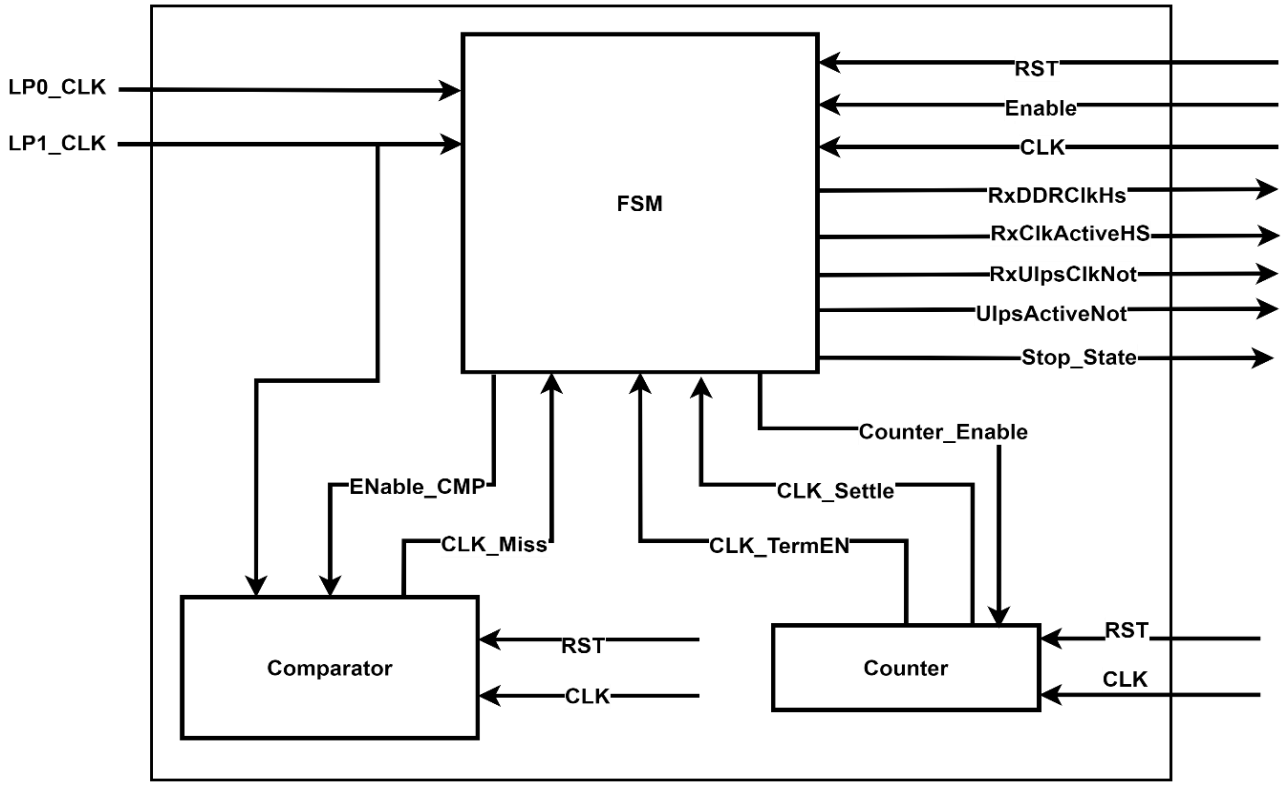


Figure 2.6: Components of CLK Lane RX block

Finite State Machine (FSM)

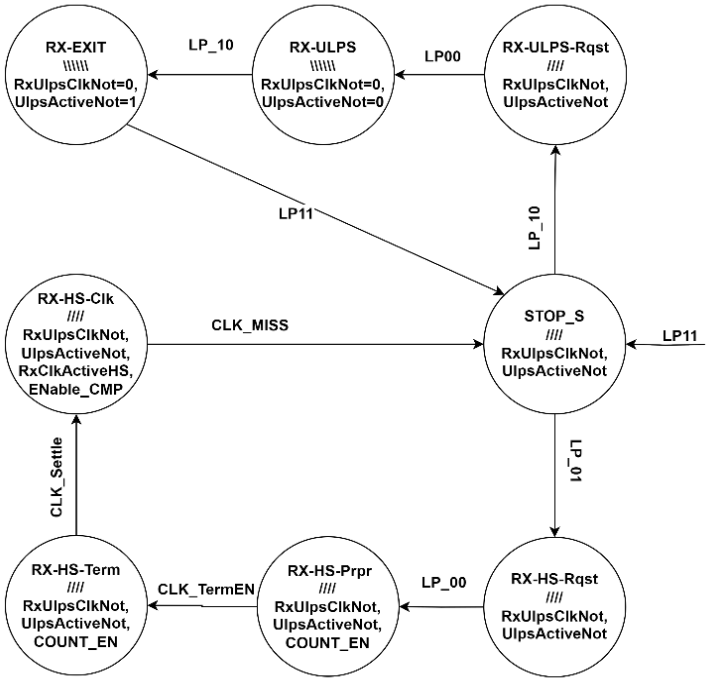


Figure 2.7: CLK Lane RX FSM block

At the heart of the Clock Lane Receiver is a finite state machine (FSM) that governs all operational states, including entry and exit from ULPS, HS prepare, stop state monitoring, and active HS clock reception. The FSM receives several control and status signals from the physical layer and protocol control logic, including:

• RxUlpsClkNot and UlpsActiveNot for monitoring ULPS state,

• COUNT_EN for controlling timing during HS transitions.

• RxClkActiveHS and Enable_CMP for detecting and validating active high- speed clock input. Based on these inputs, the FSM transitions through defined states such as RX-ULPS, STOP_S, RX-HS-Prpr, and RX-HS-Clk, managing the enablement of the HS clock path and ensuring proper synchronization. It also orchestrates the safe exit from low power states and validates incoming clock signals for reliable data sampling during DDR operations.

FSM State Descriptions:

- LP11: Idle state where both RxUlpsClkNot and UlpsActiveNot are high, indicating no active communication.
- RX-ULPS: Entered during Ultra-Low Power State where both RxUlpsClkNot and UlpsActiveNot are low.
- RX-ULPS-Rqst: Exit from ULPS begins when either RxUlpsClkNot or UlpsActiveNot becomes high.
- RX-EXIT**:** ULPS exit state where UlpsActiveNot is high while RxUlpsClkNot remains low.
- STOP_S: Stop state indicating a non-ULPS idle condition with RxUlpsClkNot high and UlpsActiveNot high.
- RX-HS-Rqst: Detects request for High-Speed reception when both RxUlpsClkNot and UlpsActiveNot are high.
- RX-HS-Prpr: Prepares for HS reception with UlpsActiveNot and COUNT_EN asserted.

- RX-HS-Term: Performs HS termination phase with active COUNT_EN.
- RX-HS-Clk: Active HS reception where RxClkActiveHS and ENable_CMP are monitored during clocking.

Timing Control (Counter Module)

A dedicated counter module is integrated to manage the precise timing requirements of the various protocol-defined phases, including:

Parameter	Description	Time
TCLK-SETTLE	Time interval during which the HS receiver should ignore any Clock Lane HS transitions, starting from the beginning of TCLK-PREPARE.	95 → 300 ns
TCLK-TERM-EN	Time for the Clock Lane receiver to enable the HS line termination, starting from the time point when Dn crosses VIL, MAX.	38 ns (MAX).

Table 2.4: Timing Parameters of Counter of CLK lane RX

The FSM enables the counter during each transition phase to ensure all signal timing parameters strictly adhere to the global timing constraints specified in Section 6.9 and Table 14 of the MIPI D-PHY specification. This guarantees interoperability and signal integrity across the link.

Comparator (CMP)

The FSM waits for the clk prepare time (300ns) for the clock to stabilize before enabling the comparator (Enable_CMP). The comparator checks for valid clock transitions while a Clock Miss Timeout runs in parallel. If no transitions are detected of received CLK within the timeout, the FSM disables the Clock Lane HS-RX to prevent errors.

Parameter	Description	Time
TCLK-MISS	Timeout for receiver to detect absence of Clock transitions and disable the Clock Lane HS-RX.	60 ns (MAX)

Table 2.5: Timing Parameters of Comparator of CLK lane RX

2.3 MIPI D-PHY TRANSMITER TOP LEVEL

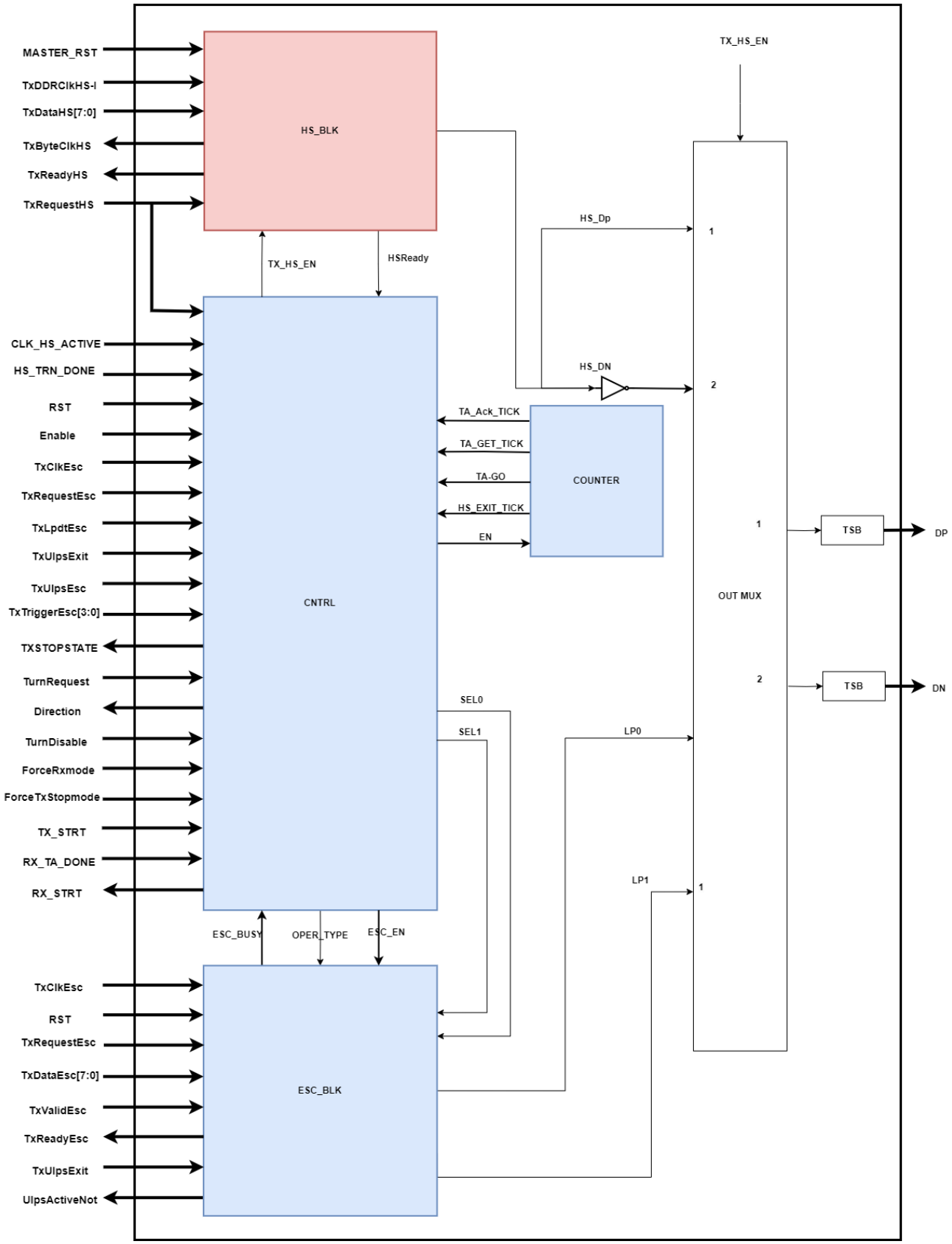


Figure 2.8: DATA Lane TX block

The figure illustrates the architecture of a **MIPI D-PHY Transmitter Lane**, which enables reliable data transmission across multiple signaling modes, including High-Speed (HS), Low-Power (LP), and Escape mode. The design integrates several coordinated blocks to handle transmission control, serialization, and mode transitions as defined by the MIPI specification.

• HS_BLK (High-Speed Block)
This block is responsible for high-speed data transmission. It receives parallel data (TxDataHS [7:0]) and a byte clock (TxByteClkHS), serializes the data, and drives it on the HS differential output (HS_Dp). The HS transmission is enabled by TxRequestHS, and the block asserts TX_HS_EN to signal activation.

• ESC_BLK (Escape Mode Block)
The Escape Block handles all low power signaling and escape mode operations, including ULPS entry and exit. It drives the single-ended low-power signals LP0 and LP1 based on control inputs such as TxRequestEsc, TxUlpsExit, and TxClkEsc. It also outputs operation type and activity status to the control unit.

• CNTRL (Control FSM)
This is the central control finite state machine that coordinates mode transitions based on external control signals (e.g., TxRequestHS, TxUlpsExit, Enable, TurnRequest). It selects whether to activate HS, LP, or Escape mode by enabling the corresponding blocks and controlling multiplexers. It also ensures protocol timing and state synchronization.

• COUNTER
A timing block that generates delay ticks required for protocol-defined timing such as Time-A, Time-B, and HS exit delays. Signals like TA_GO, HS_EXIT_TICK, and TA_Ack_TICK are used to synchronize transitions between states.

• OUT MUX (Output Multiplexer)
This multiplexer selects the final output source either HS data (HS_Dp) or LP signals (LP0, LP1) based on the active transmission mode. The selected signal is then routed to the output pad through the TSB block.

2.3.1 MIPI D-PHY TRANSMITER CONTROL UNIT

These two FSMs work together to implement the transmitter controller. Here's a simplified explanation of how they interact:

FSM 1 (Main Control) handles the overall transmission flow including:

- Lane turnaround management
- Low Power mode operations
- Escape mode for special commands
- Coordination with the receiver

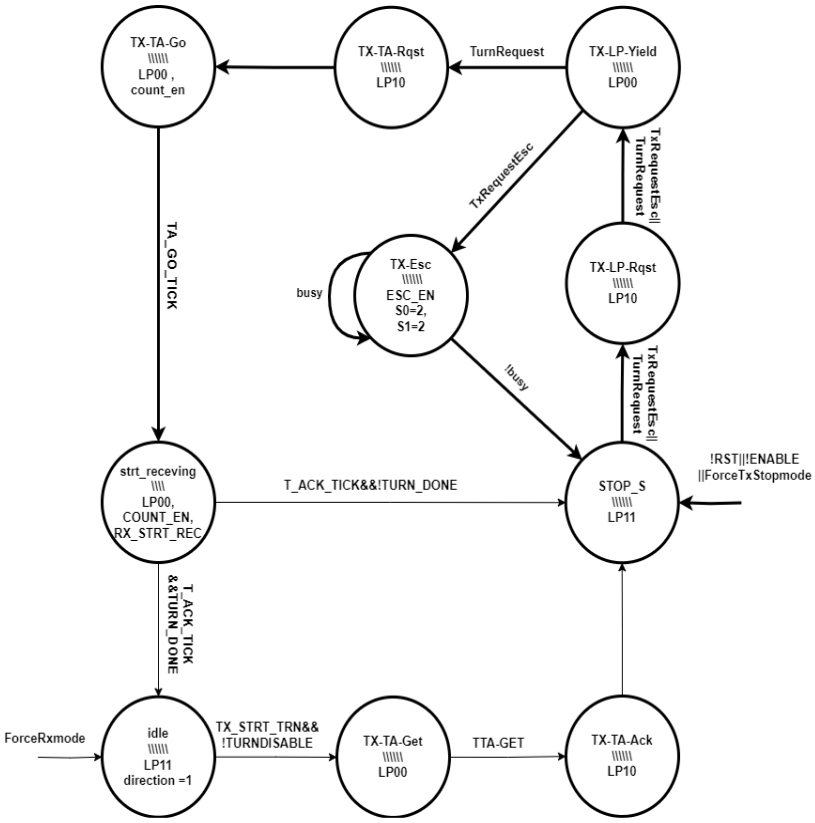


Figure 2.9: DATA Lane TX Main FSM

FSM 2 (High Speed Control) specifically manages:

- High Speed mode entry and exit
- HS transmission preparation
- HS block transmission
- Proper HS-to-LP transitions.

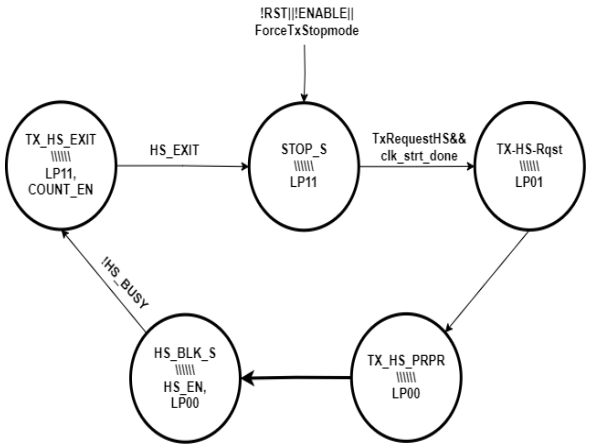


Figure 2.10: DATA Lane TX HS FSM control

FSM State Descriptions of FSM 1 – Main Control FSM (CNTRL Unit)

• Idle (LP11): Default idle state where both data lines are held in LP-11, indicating no active transmission.
• TX_LP_Rqst (LP10): Initiates a low-power transmission request by driving the lines to LP-10, signaling intent to enter escape or turnaround mode.
• TX_LP_Yield (LP00): Releases bus control and transitions the lines to LP-00, preparing for escape or turnaround handshake.
• TX_Esc (ESC_EN, S0-2, S1-2): Activates escape mode with the escape block enabled and sets the operation type, used for ULPS, trigger, or control signaling.
• TX_TA_Rqst (LP10): Signals a turnaround request by setting the lines to LP-10, requesting the right to transmit after the receiver.
• TX_TA_Go (LP00): Confirms and initiates the turnaround process by transitioning the lines to LP-00.
• strt_recving (LP00, RX_STRT_REC): Enters a receiver preparation state, holding LP-00 while the receiver prepares to begin.
• TX_TA_Get (LP00): Gains bus control during the turnaround phase while maintaining LP-00 on the lines.
• TX_TA_Ack (LP10): Acknowledges the turnaround grant and asserts LP-10 before transitioning to HS or Escape modes.

FSM State Descriptions of FSM 2 – High-Speed Control FSM

• TX_HS_Rqst (LP01): Requests entry into high-speed mode by driving LP-01, indicating the beginning of HS preparation.
• TX_HS_PRPR (LP00): Prepares for HS transmission by holding LP-00, allowing proper line conditioning before switching to HS.
• HS_BLK_S (LP00): Activates the HS block while maintaining LP-00; may be used to delay HS start until ready.
• TX_HS_Exit (LP11): Ends HS transmission and cleanly transitions the data lines back to idle LP-11.
• Stop_S (LP11): A safe stop state that ensures both data lines return to LP-11, either from HS or LP activity.

2.3.2 MIPI D-PHY HIGH SPEED TRANSMITTER (HS_TX)

The High-Speed Transmitter (HS-TX) in MIPI D-PHY is designed to transmit serialized data at high data rates using differential signaling over Data and Clock lanes. Its primary role is to send high-speed data bursts during active transmission periods. The transmitter operates in DDR (Double Data Rate) mode, where data is launched on both rising and falling edges of the high-speed clock (TxDDRClkHS). The HS-TX is activated by a TxRequestHS signal, and transitions into HS mode are managed by a finite state machine (FSM) to meet protocol timing requirements. Each lane in the transmitter supports lane escape to enter low-power modes when no data is transmitted, improving power efficiency. The transmitter works in coordination with the Clock Lane, which provides a synchronized clock for the data lanes to follow.

The proposed design of the High-Speed Transmitter is composed of five main architectural blocks: a finite state machine (CTRL), a clock divider, a timing control counter, a DDR serializer, and a output multiplexer.

2.3.2.1. Block Diagram

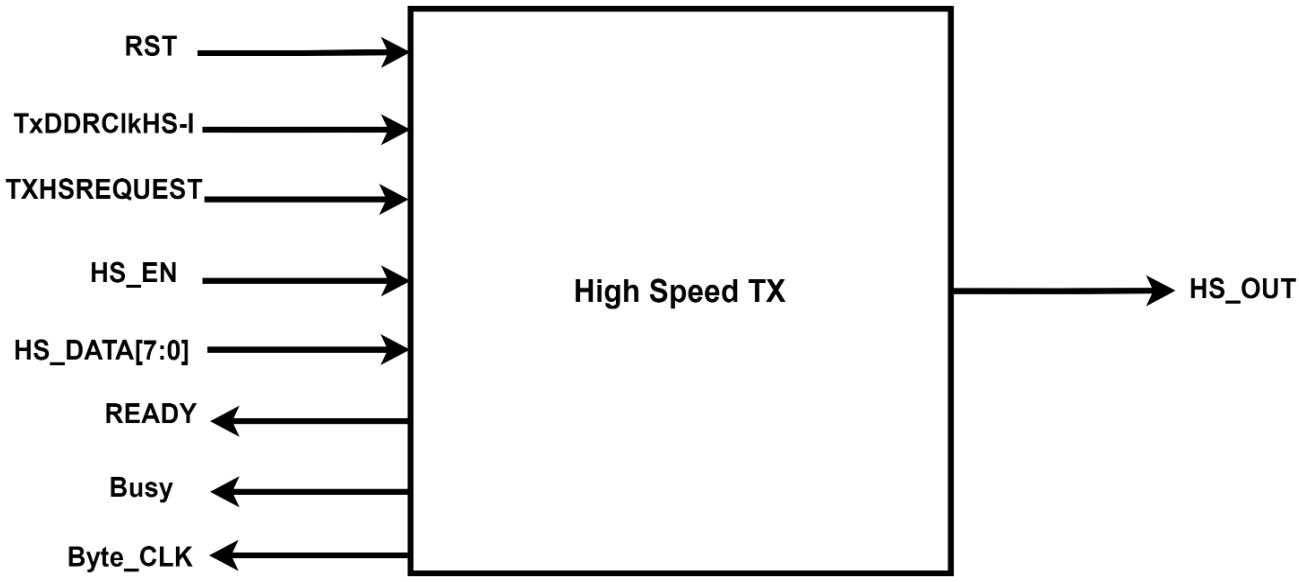


Figure 2.11: High Speed TX Block

2.3.2.2 Interface Table

Port	Direction	Width	Description
RST	Input	1-bit	High Speed TX reset
TxDDRClkHS-I	Input	1-bit	This signal is used to transmit High-Speed data bits over the Lane Interconnect
TxRequestHS	Input	1-bit	This active high signal also indicates that the protocol is driving valid data on TxDataHS to be transmitted
HS_EN	Input	1-bit	HS TX Enable controlled by FSM
HS_DATA	Input	8-bit	High-Speed Transmit Data.
Ready	Output	1-bit	This active high signal indicates that TxDataHS is accepted by the Lane Module to be serially transmitted
Busy	Output	1-bit	that signal indicates the HS Block is transmitted data
HS_OUT	Output	1-bit	Serialized data that will be transmitted
TxByteClkHS	Output	1-bit	High-Speed Transmit Byte Clock.

Table 2.6: Interface table of High-Speed TX

2.3.2.3 Architecture

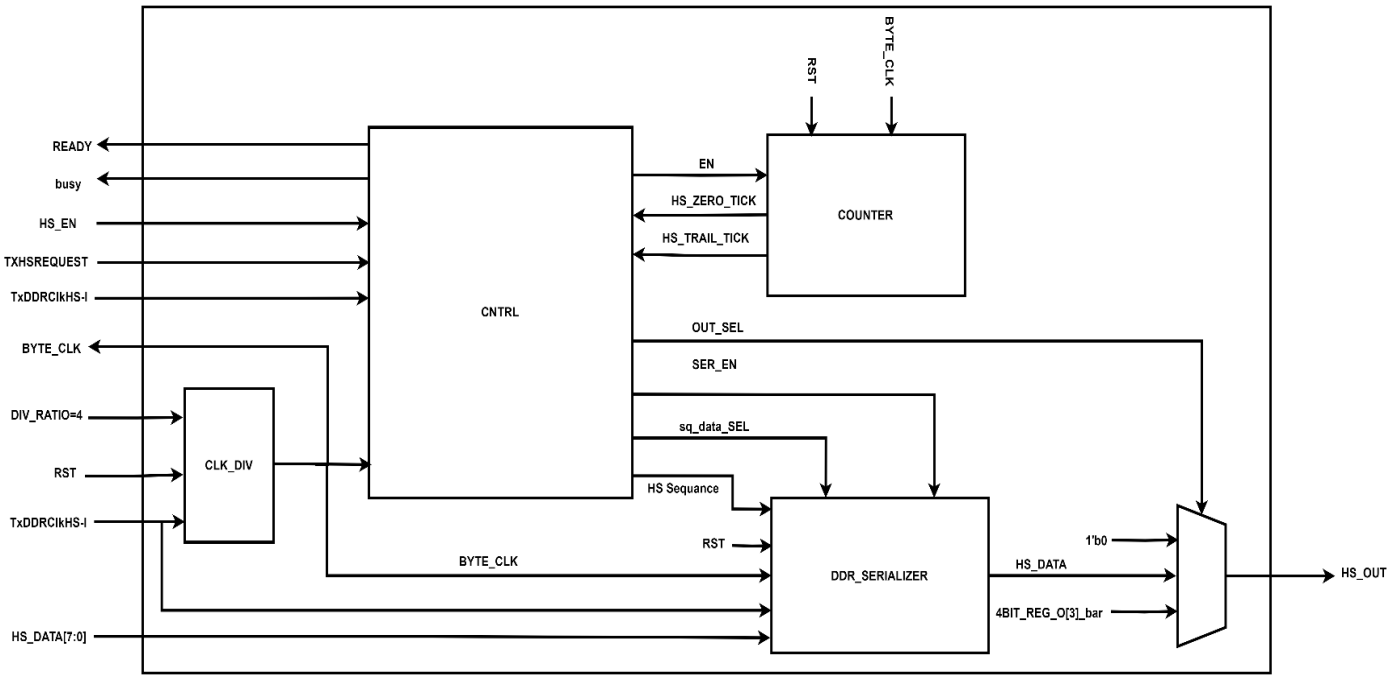


Figure 2.12: Components of HS Block

Finite State Machine (FSM)

This FSM manages the transition of the transmitter through different operational stages during High-Speed (HS) data transmission. It ensures that proper timing, signal conditioning, and synchronization are applied as per the MIPI D-PHY protocol. The HS TX FSM manages the high-speed data transmission sequence over the MIPI D-PHY Data Lane. It starts in the IDLE state, where the transmitter is inactive and waiting for HS_EN and TxRequestHS. Upon request, it transitions to TX_HS_GO, driving the line to HS-0 and enabling the byte clock and counters. After the required HS-0 period (HS_ZERO_TICK),it enters TX_HS_SYNC to align data and activate the DDR path.

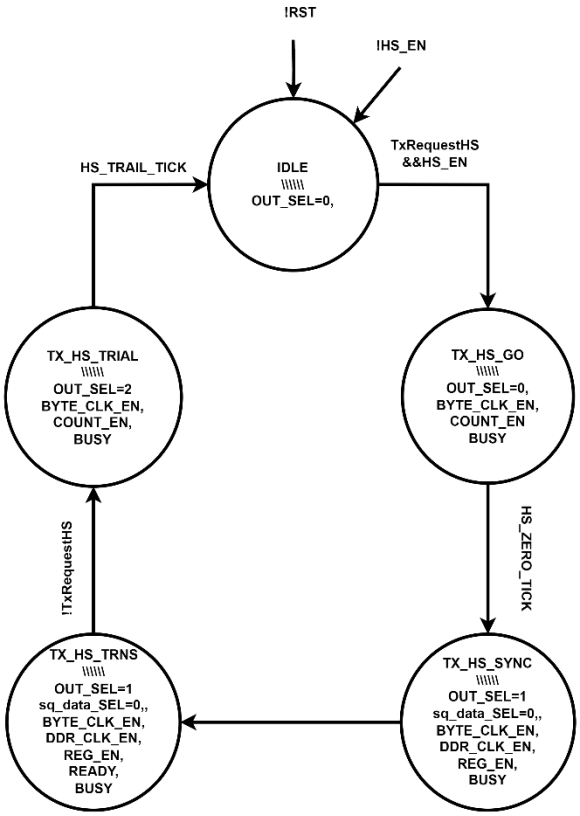


Figure 2.13: HS TX FSM

In this state, it enables DDR_CLK, output registers, and sets OUT_SEL = 1 for HS signaling. Then it moves to TX_HS_TRNS, where actual data transmission occurs with DDR signaling. The transmitter asserts READY and continues output until TxRequestHS is deasserted. On the transmission end, it moves to TX_HS_TRIAL, outputting a trailing signal (typically HS-1). OUT_SEL = 2 is set, and the FSM maintains clocking and counters for protocol compliance. After the trail timeout (HS_TRAIL_TICK), the FSM returns to IDLE, completing the HS cycle. Throughout, the FSM carefully controls clocks (BYTE_CLK, DDR_CLK) and logic enables (REG_EN). This sequence ensures timing accuracy, signal integrity, and MIPI D-PHY protocol conformance.

DDR Serializer

A DDR Serializer converts parallel data into a serial stream using both rising and falling edges of the clock, effectively doubling the data rate. In MIPI D-PHY, it enables high-speed transmission by sending 2 bits per clock cycle through a single data line.

Timing Control (Counter Module)

A dedicated counter module is integrated to manage the precise timing requirements of the various protocol-defined phases, including:

Parameter	Description	Time
THS-PREPARE	Time that the transmitter drives the Data Lane LP-00 Line state immediately before the HS-0 Line state starts the HS transmission (1 CLK of ESC CLK)	60 ns
THS-ZERO	time that the transmitter drives the HS-0 state prior to transmitting the Sync sequence.	140 ns
THS-TRAIL	Time that the transmitter drives the flipped differential state after last payload data bit of a HS transmission burst	80 ns

Table 2.7: Timing Parameters of Counter of High-Speed TX

The FSM enables the counter during each transition phase to ensure all signal timing parameters strictly adhere to the global timing constraints specified in Section 6.9 and Table 14 of the MIPI D-PHY specification. This guarantees interoperability and signal integrity across the link.

Clock Divider

This block is used to divide input clk (TxDDRClkHS-I) by 4 to generate (TxByteClkHS). This is used to synchronize PPI signals in the High-Speed transmit clock domain. It is recommended that all transmitting Data Lane Modules share one TxByteClkHS signal. The frequency of TxByteClkHS is exactly 1/8 the High-Speed bit rate.

Output Multiplexer (MUX)

This MUX is used to select the output of HS block between 3 options

1- Select 1’b0 when drive the lane to transmit HS-0 state prior to transmitting the Sync sequence.
2- Select HS data when sending the HS sequence or the HS Data on the lane after serializing it.
3- Select the negative last bit of payload data and transmit it on the lane for end of transmission HS for a time THS-TRAIL then Disables the HS-TX.

2.3.3 MIPI D-PHY ESCAPE MODE

The TX_ESC block is a critical submodule within the TX control unit, managing the transmission of escape mode sequences. It interfaces with a WAKEUP_COUNTER, a multiplexer, and an ESC_SERIALIZER to process and serialize escape mode data. Its primary role is to handle special control signals (e.g., ULP entry/exit, trigger sequences).

Functionality and Importance

- **Escape Mode Handling**: The TX_ESC block supports MIPI D-PHY escape modes, such as ULP (Ultra-Low-Power) entry/exit and trigger commands, which are essential for low-speed control and power management.
- **Sequence Control**: It manages predefined sequences (LP0T_SEQ, ULP_SEQ, TRIGGER_SEQ) for specific operations (e.g., wake-up, calibration), triggered by TxTriggerEsc [3:0].
- **Data Transmission**: Processes 8-bit data (TxDataEsc [7:0]) when TxValidEsc is active, ensuring proper serialization and readiness acknowledgment via TxReadyEsc.

2.3.3.1. Block Diagram

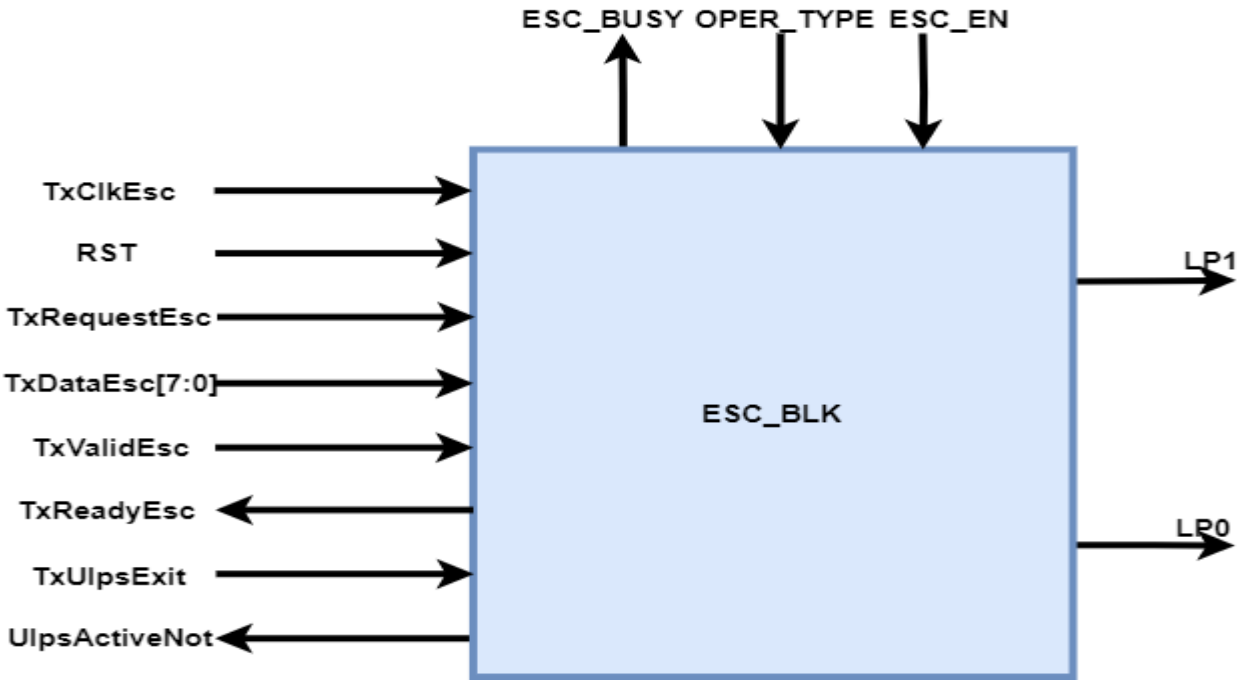


Figure 2.14: Escape Block TX

2.3.2.2 Interface Table

Port	Direction	Width	Description
RST	Input	1-bit	TX reset
TxClkEsc	Input	1-bit	This signal is used to transmit low-power data bits over the Lane Interconnect
TxRequestHS	Input	1-bit	Escape mode Transmit Request
TxDataEsc	Input	8-bit	Low-Power Transmit Data
TxValidEsc	Input	1-bit	Escape mode Transmit Data Valid
TxReadyEsc	Output	1-bit	Escape mode Transmit Ready
TxUlpsExit	Input	1-bit	Transmit ULP Exit Sequence
UlpsActiveNot	Output	1-bit	ULP State (not) Active
ESC_EN	Input	1-bit	ESC TX Enable controlled by FSM
Oper_type	Input	3-bit	That signal indicates the type of operation that the escape block will perform (LPDT, ULP And trigger)
Esc_Busy	Output	1-bit	that signal indicates the escape block is transmitted data
LP0, LP1	Output	1-bit	Serialized data that will be transmitted

Table 2.8: Interface Table of Escape Block TX

2.3.2.3 Architecture

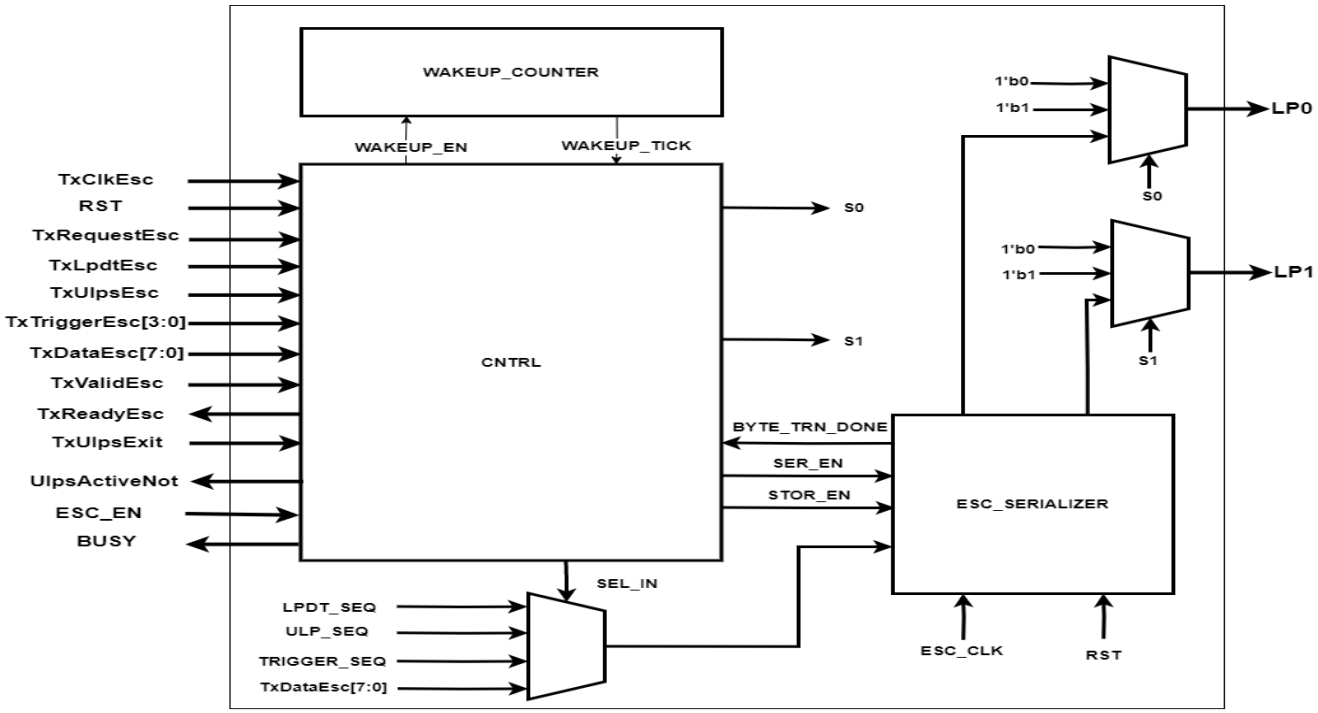


Figure 2.15: Components of ESC Block

Control Unit

The CNTRL block acts as the brain of the TX_ESC system, managing the logic and state transitions for escape mode operations. It orchestrates the flow of control commands, ULP signaling, and trigger sequences, ensuring that the system adheres to the MIPI protocol. By coordinating the overall process, it facilitates seamless mode switches and maintains system integrity, playing a key role in achieving reliable performance across various operational states.

FSM State Descriptions

• Idle (LP11): Initial state with both data lines in high LP-11 mode, awaiting an escape mode request.
• TX_ESC_RQST (LP01): Requests escape mode entry, setting lines to LP-01 to initiate the process.
• TX_ESC_GO (LP00): Activates escape mode by driving lines to LP-00, enabling data transmission.

FSM 1 – LPDT Control FSM

• TX_LPDT_SEQ (LP00):
Executes low-power data transmission sequence while maintaining LP-00.

• TX_LPDT_TRN (LP00):
Transfers low-power data, preparing for completion acknowledgment.

• TX_MARK (LP00):
Marks the escape sequence boundary, signaling a specific protocol.

• TX_LPDT_READY (LP00): Confirms readiness to receive or process the next data byte in LP-00.

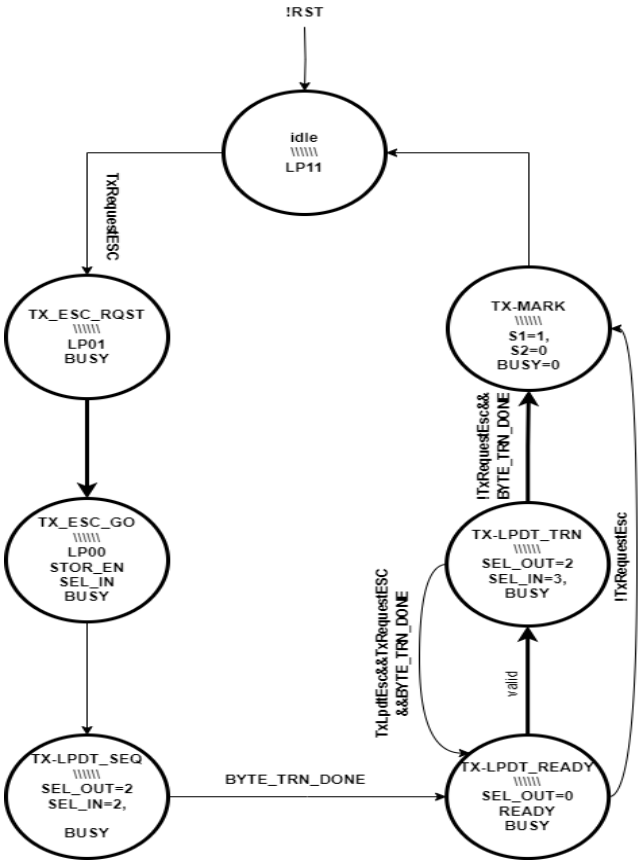


Figure 2.16: ESC mode LPDT

FSM 2 – Trigger Control FSM

• TX_TRIG (LP00): Executes the trigger command sequence while in LP-00.

• TX_TRIG_DUMMY (LP00): Serves as a placeholder state for trigger initialization in LP-00.

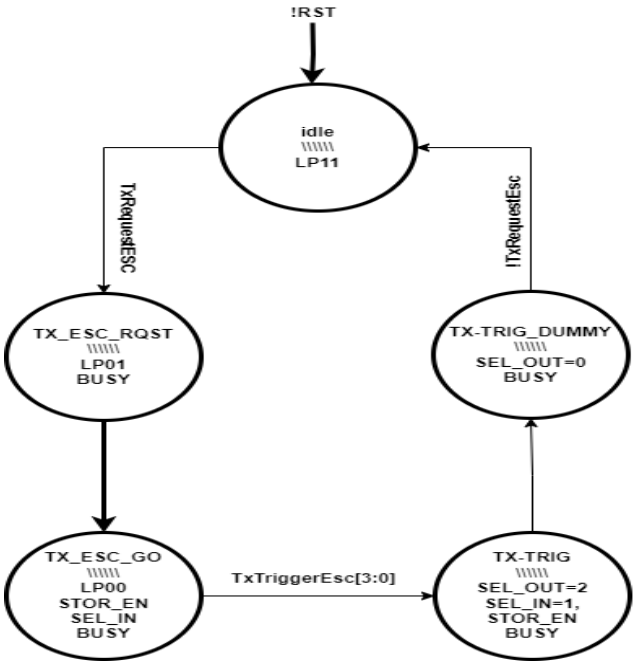


Figure 2.17: Esc mode Trigger

FSM 3 – ULPS Control FSM

• TX_ULPS_SEQ (LP00): Executes the ULP sequence ensuring proper entry or exit protocol.

• TX_ULPS (LP00): Enters ultra-low-power state, maintaining LP-00 for minimal power consumption.

• TX_MARK (LP01): Marks a specific point in the escape sequence, indicating a wake-up or transition phase.

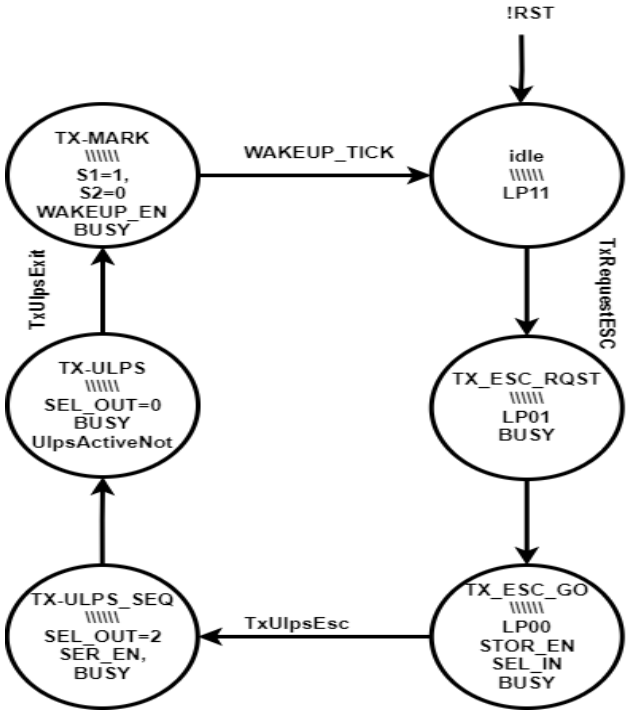


Figure 2.18: Esc mode ULPS Control

Timing Control (WAKEUP_COUNTER Module)

A dedicated counter module is integrated to manage the precise timing requirements of the various protocol-defined phases, including:

Parameter	Description	Time
TWAKEUP	Time that a transmitter drives a Mark-1 state prior to a Stop state to initiate an exit from ULPS.	1ms

Table 2.9: Timing Parameters of Counter of Init ESC mode

The FSM enables the counter during each wake-up phase to exit from ultra-low-power state, all signal timing parameters strictly adhere to the global timing constraints specified in Section 6.9 and Table 14 of the MIPI D-PHY specification. This guarantees interoperability and signal integrity across the link.

Multiplexer

The Multiplexer block in the TX_ESC module is tasked with selecting the appropriate data input for transmission based on the control signal SEL_IN from the CNTRL block. It handles multiple input options, including predefined sequences like LP0T_SEQ (low-power transition), ULP_SEQ (ultra-low-power mode), TRIGGER_SEQ (trigger commands), and raw data TxDataEsc[7:0]. This selection process is crucial for routing the correct signal to the ESC_SERIALIZER.

ESC_SERIALIZER

The ESC_SERIALIZER block is responsible for converting the selected data from the Multiplexer into serialized output signals for the LP0 and LP1 data lines in the TX_ESC module. It operates under the control of ESC_CLK (clock signal), SER_EN and STOR_EN (enable and store commands from CNTRL), and RST (reset). This block ensures that the data, whether a sequence or raw bytes, is properly formatted and timed for transmission, adhering to the protocol's low power signaling requirements. Its role is critical in driving the physical lines with binary states (1'b0 or 1'b1), enabling reliable communication of escape mode commands, ULP transitions, and trigger signals. This functionality supports efficient power management and seamless operation in low-power data applications up to 10 Mbps.

2.4 MIPI D-PHY RECIEVER DATA LANE TOP LEVEL

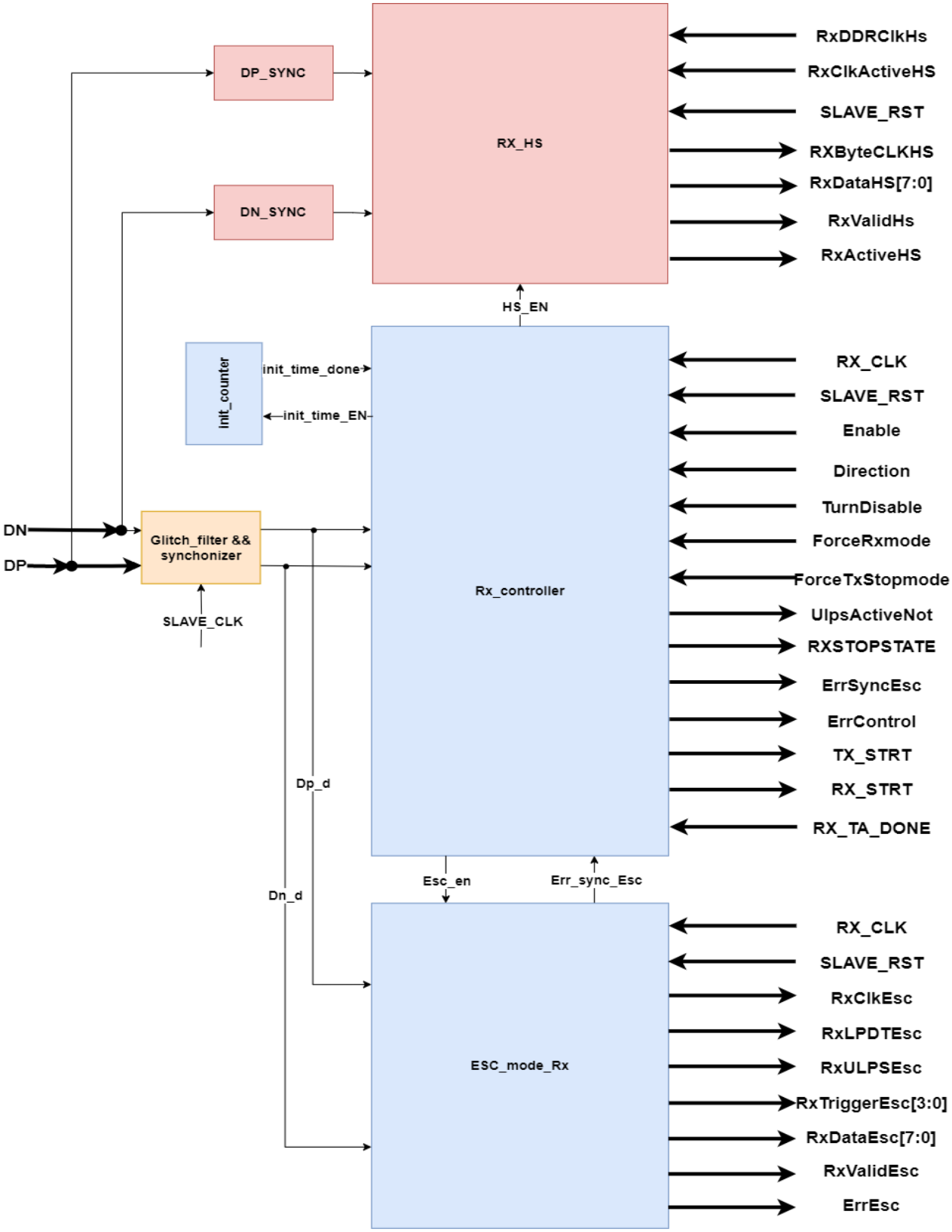


Figure 2.19: MIPI D-PHY Receiver data lane block diagram

The system depicted in the diagram represents the receiver (RX) architecture of a MIPI D-PHY v1.2 design, a high-speed interface commonly used in mobile devices for cameras and displays. At its core, the system comprises three main blocks: RX_HS, Rx_controller, and ESC_mode_Rx, each playing a critical role in handling high-speed (HS) and low-power (LP) data reception. The RX_HS block processes high-speed data from the DP and DN lines, synchronized through DP_SYNC and DN_SYNC modules, ensuring robust signal integrity at rates up to 2.5 Gbps.

RX_CONTROLLER

The Rx_controller serves as the central decision-making unit, managing mode transitions, error detection, and control signals, while interfacing with an initialization counter to coordinate startup sequences.

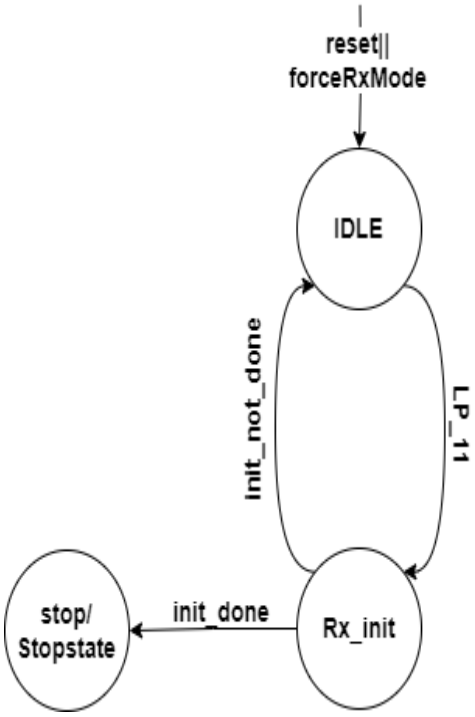


Figure 2.20: Initialization Slave

FSM Main functions:

1. Initialization
2. Manage high speed mode
3. Manage escape mode
4. Manage turn around operation

1. Initialization

After power-up, the Slave side PHY initializes when the Master PHY maintains a Stop State (LP-11) for a duration exceeding TINIT, marking the start of the Initialization period. The Master PHY is initialized via a system or Protocol signal (PPI), ensuring no Stop State longer than TINIT occurs before its initialization, while the Slave ignores all line states prior to this period. In multi-Lane setups, all lanes initialize simultaneously. with TINIT being a protocol-dependent parameter having a minimum of 100 µs for both Master and Slave, though specific values (e.g., ≥1 ms for Master, 500-800 µs for Slave) may be defined by the protocol layer.

Timing Control (Counter Module)

A dedicated counter module is integrated to manage the precise timing requirements of the various protocol-defined phases, including:

Parameter	Description	Time
TINIT	TINIT being a protocol-dependent parameter defined by the protocol layer	>100 µs

Table 2.10: Timing Parameters of Init Slave

FSM description

- **IDLE**: Initial state where the receiver awaiting LP-11.
- **Rx_init**: Active state where the receiver performs initialization, guided by a external init_time signal until completion.
- **Stop_state** (LP11): State with both data lines in high LP-11 mode, awaiting requests.

2. Manage high speed mode

After Initialization, the Slave side PHY receive data from master side, high speed data is type of this data and the Slave receive data at 200 MHz speed (400 Mbps data rate)

FSM description

- **IDLE**: Initial state where the receiver awaiting LP-11.
- **Rx_init**: Active state where the receiver performs initialization, guided by an external init_time signal until completion.
- **Stop_state** (LP11): State with data lines in high LP-11 mode, awaiting requests.

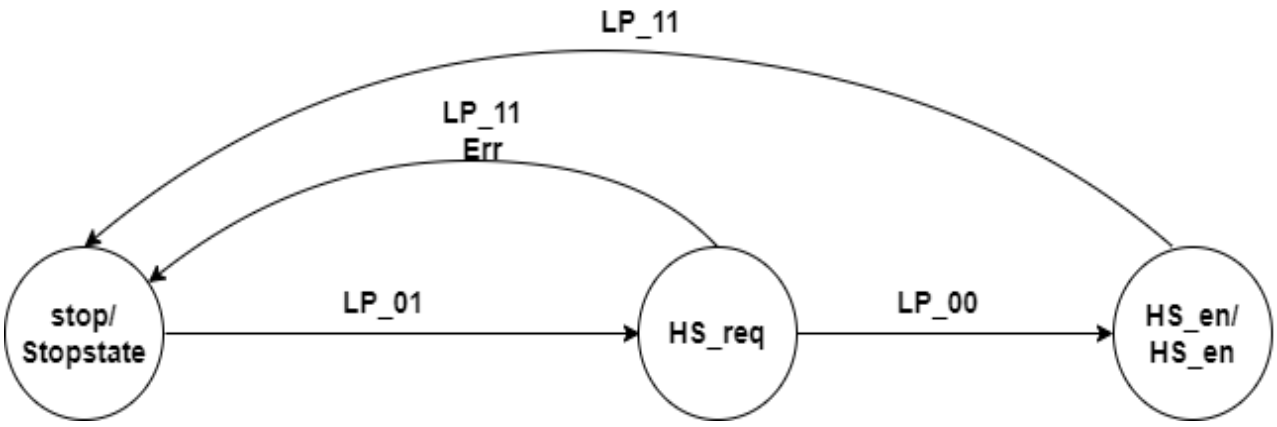


Figure 2.21: HS RX Enter control FSM

3. Manage Escape mode operation

The finite state machine (FSM) controls escape mode logic in a MIPI D-PHY v1.2 transmitter, managing low-power (LP) state transitions, escape initiation, and error handling. It includes key states:

• **Stop/StopState**: Initial idle state, transitioning to LP-11 or LP-11_Err based on system conditions.
• **LP-11**: Moving to LP_req on request or LP-11_Err on error.
• **LP-11_Err**: Handles faults in LP-11 and returns to stop or LP_req after recovery.
• **LP_req**: Requests mode change; transitions to LP-00 or LP-01 based on Esc_en.
• **LP-00**: Intermediate state for turnaround or escape entry, leading to LP_yield.
• **LP-01**: Prepares for escape; transitions to Esc_req when enabled.
• **LP_yield**: Finalizes turnaround and returns to LP-00 or other states.
• **Esc_req**: Starts the escape sequence, then returns to LP-11.

This FSM enables proper escape handling, error recovery, and LP signaling coordination

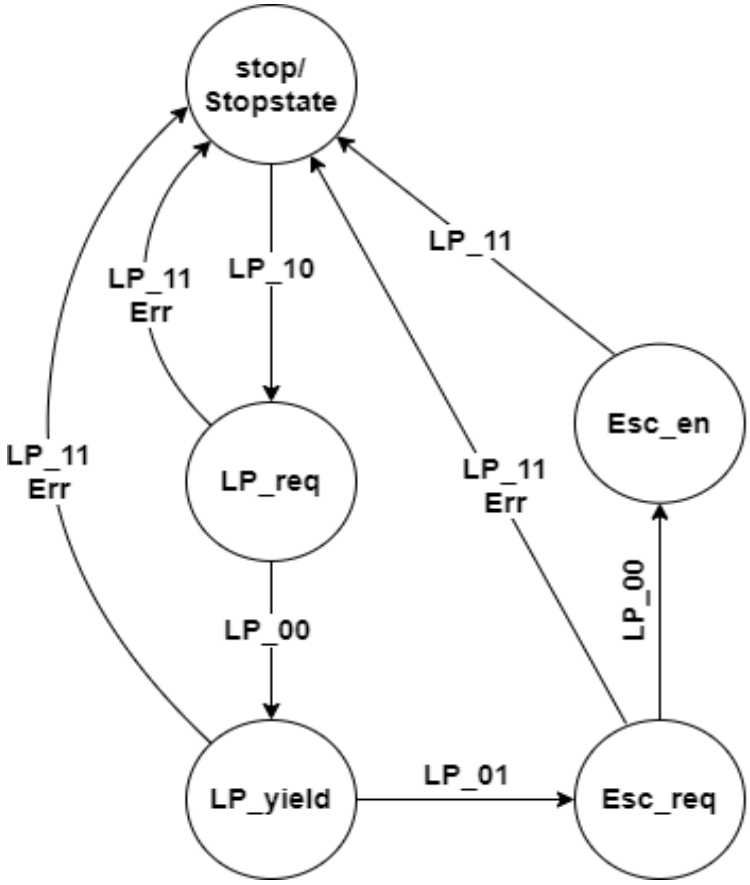


Figure 2.22: Escape mode control FSM

4. Manage turn around operation

The finite state machine (FSM) manages transitions between low-power states during reception and turnaround in MIPI D-PHY:

- **LP-11** is the default idle state. A start request triggers a move to **Ack**.
- **Ack** handles acknowledgment, loops on valid reception, or moves to stop or error states.
- **AckLook** monitors for acknowledgment signals and proceeds to **Wait**.
- **Wait** prepares for turnaround and moves to **LP-00** upon transmission start.
- **LP-11_Err** signals error recovery and transitions to **LP_req**.
- **LP_req** initiates turnaround via **LP-10** and proceeds to **LP-00**.
- **LP-00**, **LP-10**, and **LP_yield** complete the turnaround by yielding control and returning to idle or prepare states.

This FSM ensures proper role-switching between transmitter and receiver while maintaining protocol compliance.

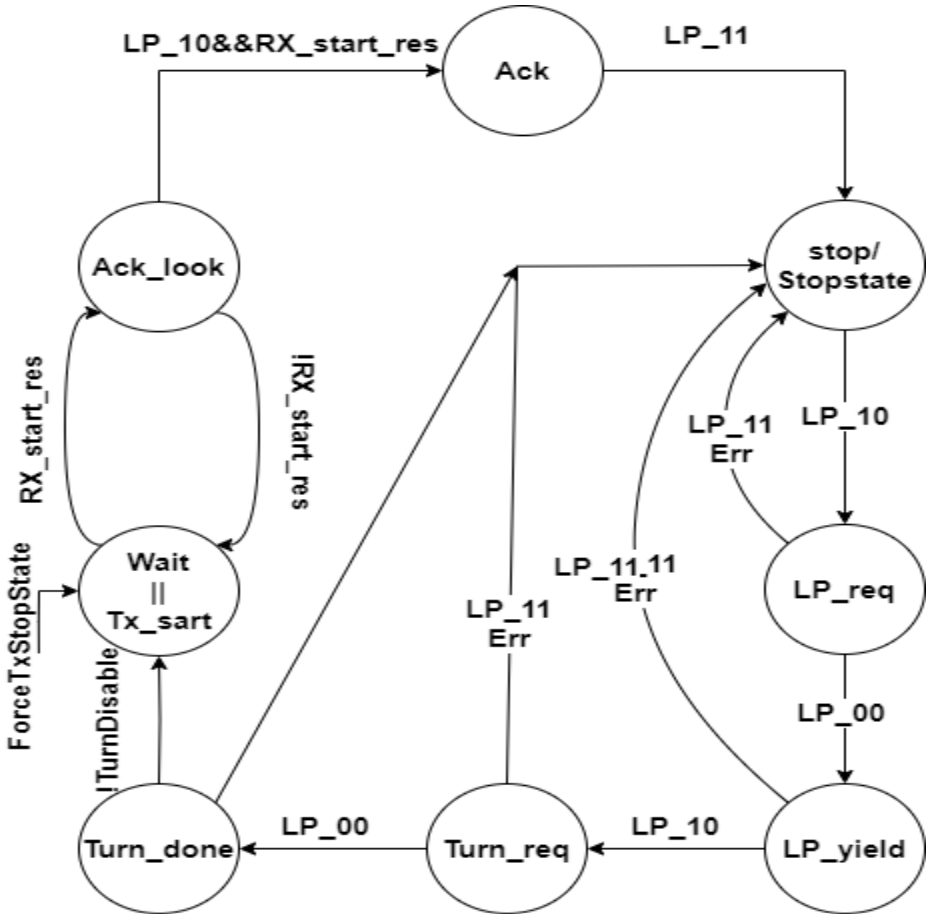


Figure 2.23: Turnaround control FSM

2.4.1 MIPI D-PHY HIGH SPEED RECEIVER (HS_RX)

The High-Speed Receiver (HS-RX) in MIPI D-PHY is designed to receive serialized high-speed data from the differential Data Lanes during active transmission. It operates in DDR (Double Data Rate) mode, capturing data on both the rising and falling edges of the incoming high-speed clock (DDRClkHS) provided by the Clock Lane. The HS-RX is enabled when a valid clock is detected and transitions into HS mode are managed by a finite state machine (FSM), which monitors the Clock Lane's prepare, zero, and sync phases to ensure timing compliance. Once in HS mode, the receiver activates its internal deserialization and data sampling circuits to recover the byte-aligned data stream. It includes timeout logic to detect clock loss or end of transmission, allowing safe exit from HS mode into low-power or idle states. The HS-RX operates in coordination with the Clock Lane and system FSM to ensure robust, synchronized, and protocol-compliant data reception.

The proposed design of the High-Speed Transmitter is composed of six main architectural blocks: a finite state machine (CTRL), a clock divider, a detector, a DDR deserializer, a trail counter and a Synchronous FIFO.

2.4.1.1. Block Diagram

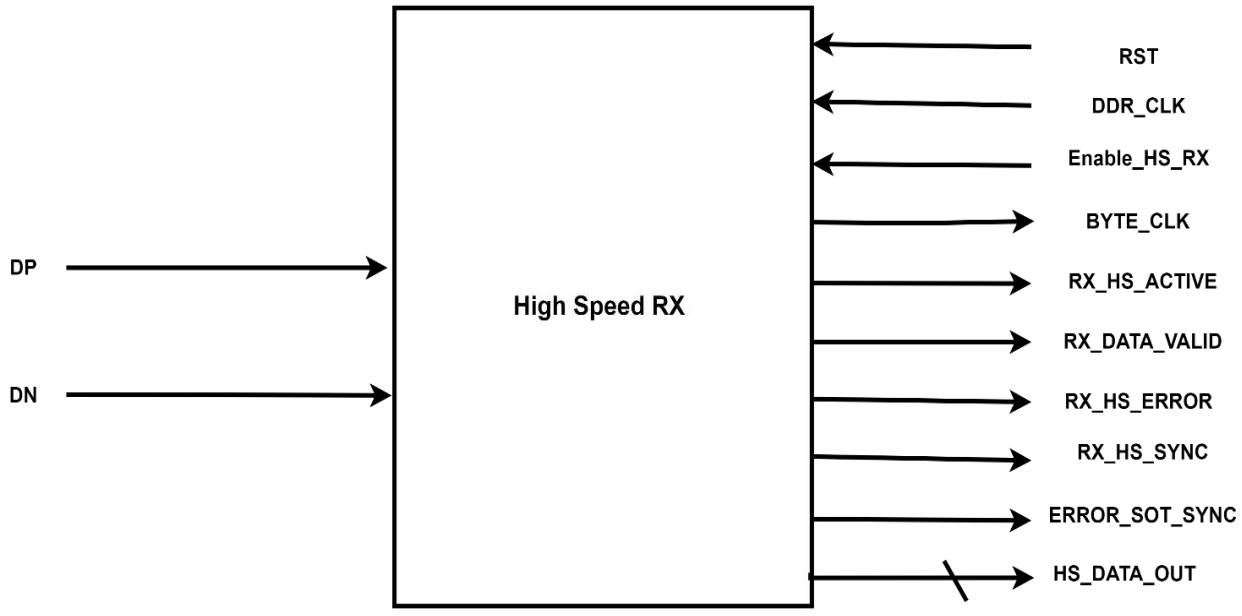


Figure 2.24: High Speed RX Block Diagram

2.4.1.2. Interface Table

Port	Direction	Width	Description
Dp, Dn	Input	1-bit	Received Serial data input from TX
RST	Input	1-bit	High Speed RX reset
DDR_Clk	Input	1-bit	Clock of HS RX this clock received from CLK lane
Enable_HS_RX	Input	1-bit	HS RX Enable controlled by FSM
HS_DATA_OUT	Output	8-bit	High-Speed Received Data (Parallel).
RX_HS_Active	Output	1-bit	This active high signal indicates that the Lane Module is actively receiving a HS transmission from the Lane.
RX_Data_Valid	Output	1-bit	This active high signal indicates that the Lane Module is driving data to the protocol on the RxDataHS output.
RX_HS_SYNC	Output	1-bit	This active high signal indicates that the Lane Module has seen an appropriate synchronization event. (1 cycle high).
RX_HS_ERROR	Output	1-bit	Start-of-Transmission (SoT) Error. If SoT leader sequence is corrupted, but in such a way that proper synchronization can still be achieved, (1 cycle high).
ERROR_SOT_SYNC	Output	1-bit	Start-of-Transmission Synchronization Error. If the SoT leader sequence is corrupted in a way that proper synchronization cannot be expected (1 cycle high)
Byte_Clk	Output	1-bit	High-Speed Received Byte Clock.

Table 2.11: Interface Table of High-Speed RX

2.4.1.3 Architecture

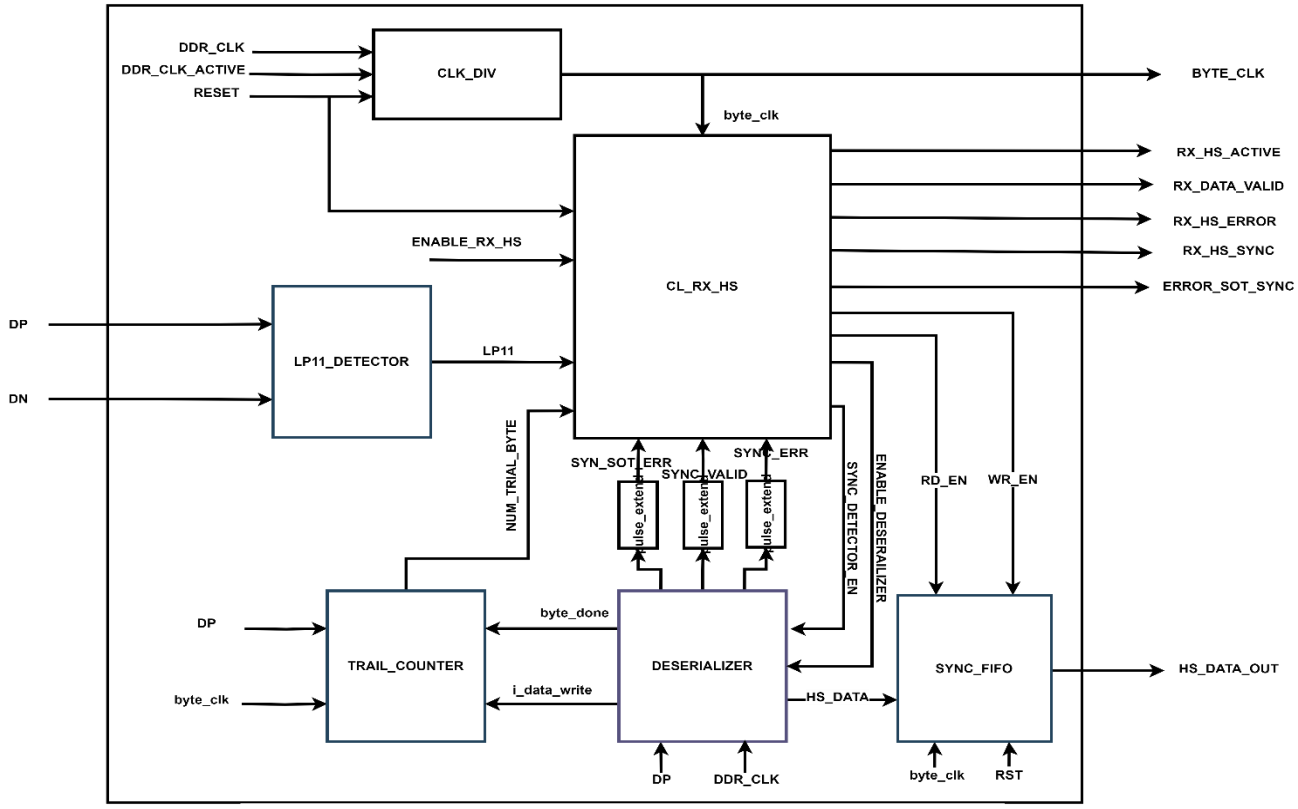


Figure 2.25: Components of HS RX Block

Finite State Machine (FSM)

This FSM governs the behavior of the HS Receiver in a MIPI D-PHY interface. It begins in the IDLE state and transitions occur based on control and error signals. When high-speed data transfer is enabled (HS_EN), the FSM moves to SYNC_CHECK to verify synchronization. If synchronization is successful (DESER_EN=1), it transitions to SYNC_VALID_STATE where the deserializer is enabled and data counting begins. Upon reaching a count of 6, the FSM enters HS_ACTIVE, indicating that valid high-speed data reception has started. The FSM monitors for completion

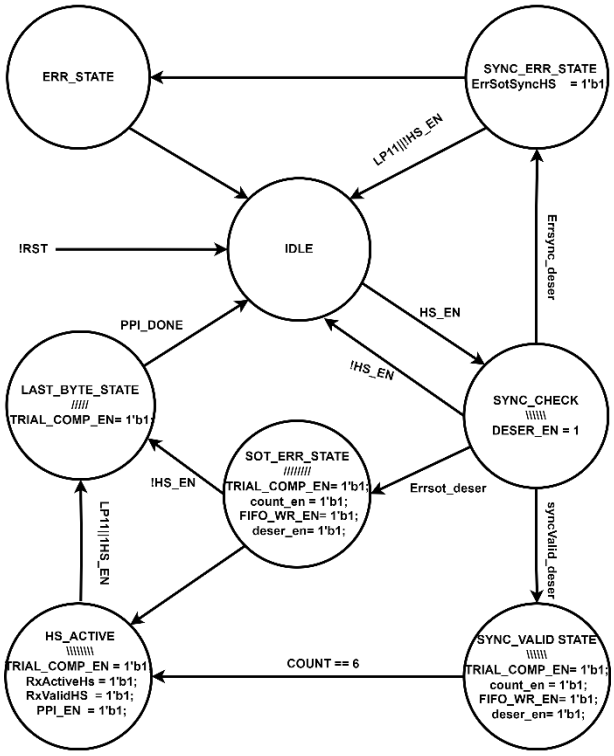


Figure 2.26: HS RX FSM

of the current data packet (PPL_DONE) or detects the last byte to transition to LAST_BYTE_STATE. Trial comparison enables are used to confirm data completion before returning to IDLE. Errors during synchronization route the FSM to SYNC_ERR_STATE, while deserializer-related errors go to SOT_ERR_STATE. Additional errors in the data path or state transitions lead to ERR_STATE. From any error state, the FSM can reset back to IDLE via (! RST) signal. The design ensures robust recovery from sync loss or data corruption.

Clock Divider

This block is used to divide input clk (DDR_Clk) by 4 to generate (Byte_Clk). This is used to synchronize signals in the High-Speed receive clock domain.

DDR Deserializer

A DDR deserializer captures serial data on both clock edges, effectively doubling data rate. In MIPI D-PHY HS RX, it aligns incoming serial bits using sync patterns and converts them into parallel words. It works with control signals like DESER_EN and FIFO_WR_EN to ensure valid data transfer. The deserializer also detects alignment or SOT errors and cooperates with the FSM for reliable high-speed data reception.

Detector (LP-11)

Detection of the LP-11 state indicates that both Dp and Dn lines are high, signaling the end of an HS burst. It marks the transition back to Low-Power mode, meaning the transmitter has completed HS data transmission. The receiver uses LP-11 detection to safely exit HS RX mode and prepare for the next command or idle state.

Trail counter

The trail counter counts the number of extra clock cycles or bytes received after the last valid data byte in a High-Speed transmission. It monitors the line to detect when the LP-11 state is reached, signaling the true end of the burst. The counter ensures these extra bytes are not misinterpreted as valid data. Once the expected trail length is reached or LP-11 is detected, the receiver finalizes the packet.

Synchronous FIFO

A synchronous FIFO is a memory buffer that stores incoming high-speed (HS) data in the same clock domain, ensuring synchronized timing between the deserializer and the protocol interface. In MIPI D-PHY HS RX, it plays a crucial role in isolating valid data from unwanted bytes like trail data (extra bytes received after valid payload ends but before LP-11 is detected).

During HS transmission, the deserializer outputs a continuous stream of data, including a few trailing bytes due to pipelining and timing overlap. However, these extra bytes are not part of the actual payload and should not be forwarded to the protocol layer.

The FSM and trail counter work together to detect when valid transmission ends. At that point, the control logic disables the FIFO_WR_EN signal, stopping writing to the FIFO. Because FIFO only captures data when FIFO_WR_EN is active, the invalid trailing bytes are blocked from entering the FIFO.

2.4.2 Escape Mode Top Level

Escape mode is a key feature in the MIPI D-PHY protocol that enables the receiver to operate at low power and handle special signaling conditions. Unlike high-speed (HS) transmission, which uses differential signaling at high frequencies, escape mode operates in low-power (LP) mode using single-ended signaling on the Dp and Dn lines.

In escape mode, the receiver detects specific line states and sequences to interpret asynchronous control commands such as:

- **ULPS (Ultra-Low Power State)** entry and exit
- **Trigger events**
- **LPDT**

2.4.2.1. Block Diagram

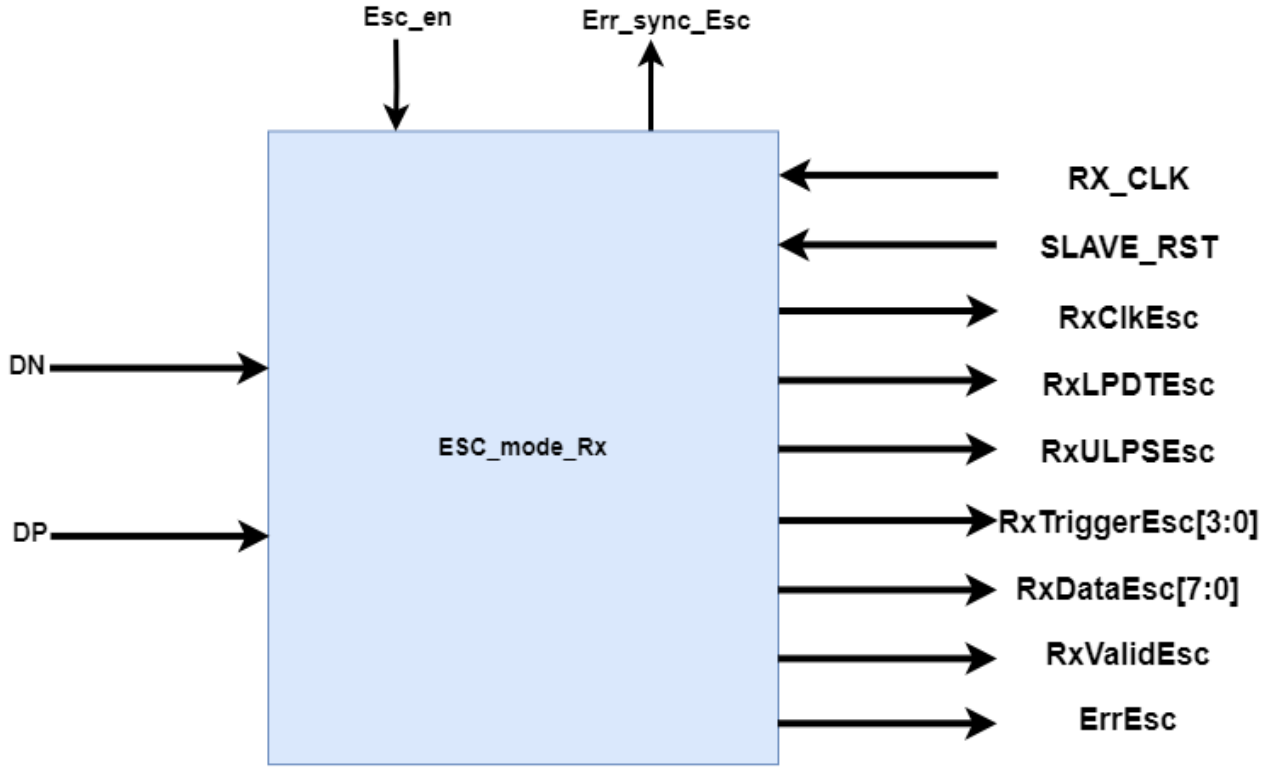


Figure 2.27: Escape RX Block

2.4.2.2. Interface Table

Port	Direction	Width	Description
Dp, Dn	Input	1-bit	Received Serial data input from TX
RST	Input	1-bit	RX reset
Enable_HS_RX	Input	1-bit	ESC RX Enable controlled by FSM
Err_Sync_Esc	Output	1-bit	Error signal is flagged when there is faults like non multiple of byte data is recieved
RX_Clk	Input	1-bit	Clock of RX divided by 5
RxClkEsc	Output	1-bit	Escape Received Clock. Generated by XORing Dp and Dn
RxLpdtEsc	Output	1-bit	Flagged when the data lane entered in LPDT mode
RxUlpsEsc	Output	1-bit	Flagged when the data lane entered ULPS mode
RxTriggerEsc	Output	4-bit	Escape Mode Receive Trigger 0-3. These active high signals indicate that a trigger event has been received.

RxDataEsc	Output	8-bit	Is the byte data sent in LPDT mode
RxValidEsc	Output	1-bit	Is flagged for a one clock cycle when the data is prepared in a boundary of byte and is ready to be sent to the PPI. (1 cycle high).
ErrEsc	Output	1-bit	Escape Entry Error. If an unrecognized escape entry command is received, this active high signal is asserted and remains asserted until the next change in line state.

Table 2.12: MIPI D-PHY receiver escape mode signals

2.4.2.3 Architecture

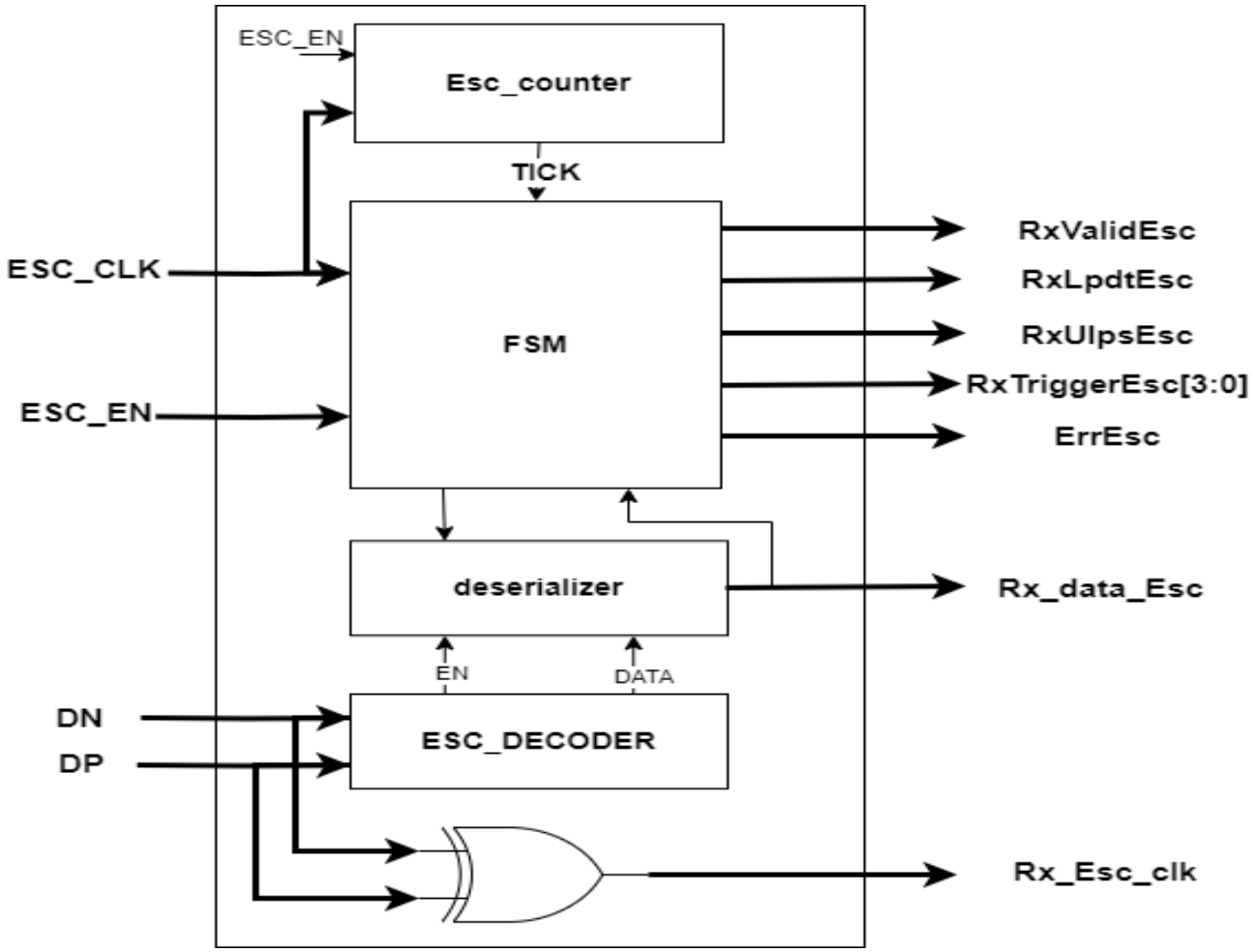


Figure 2.28: Components of ESC RX Block

2.4.2.4 Control Unit (FSM)

Control unit is Finite state machine (FSM) to follow up the input and jump from state to another state according to the entry command.

Escape Mode Action	Escape Mode Action	Entry Command Pattern
Low-Power Data Transmission	Mode	11100001
Ultra-Low Power State	Mode	00011110
Undefined-1	Mode	10011111
Undefined-2	Mode	11011110
Reset-Trigger[Remote Application]	Trigger	01100010
Unknown-4	Trigger	00100001
Unknown-5	Trigger	10100000

Table 2.13: Escape mode Pattern Command

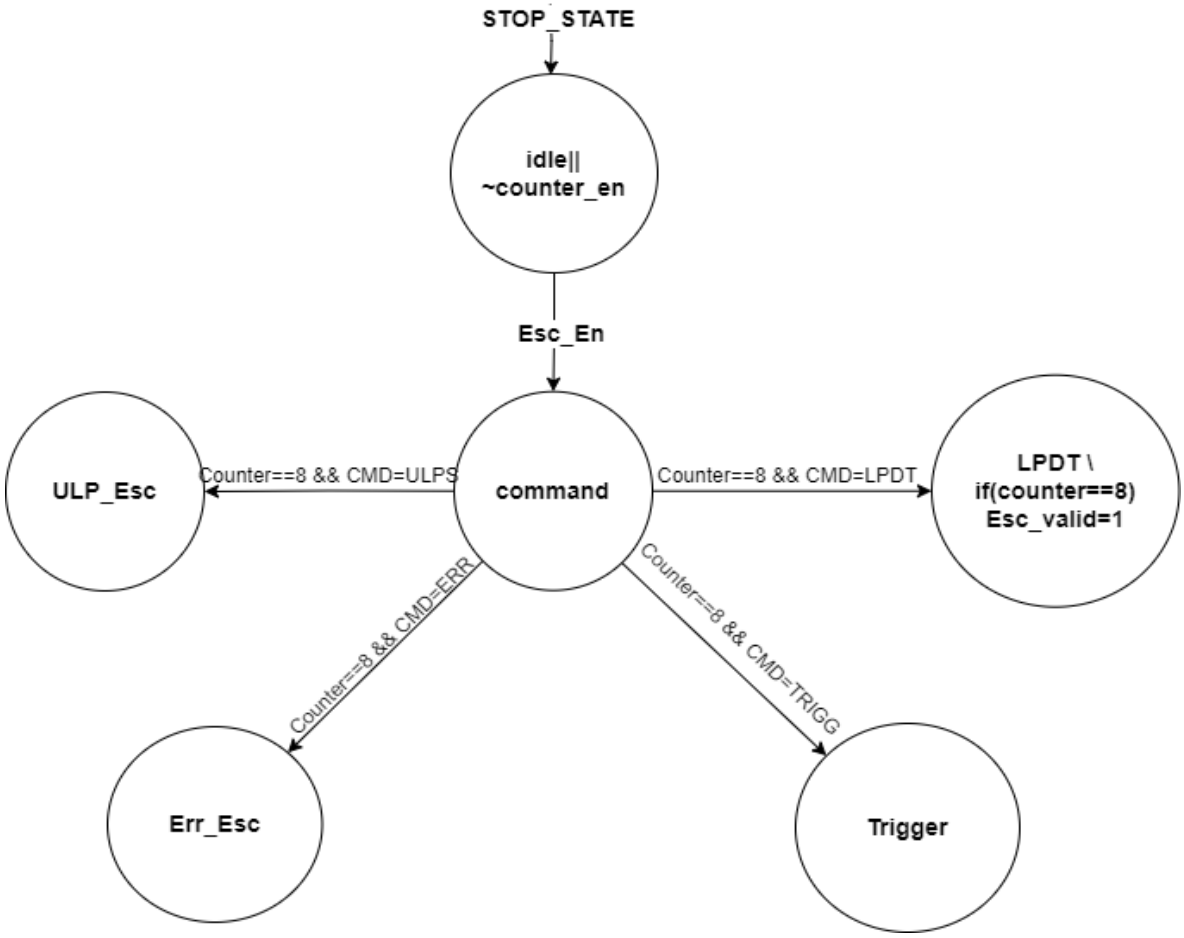


Figure 2.29: Control Unit of Escape mode

Also checks the fault in low power data transmission. we have 2 types of errors, 1st is when the transmitted data is not multiple of 8 bits or byte, 2nd is when the entry sequence not entered in the true way like when the receiver received LP01 then LP11 or LP 00 then LP11 (hint! The true way to enter the stop state LP10 (mark-1) then LP11.

DESERIALIZER

Deserializer is used to deserialize the serial transmitter data and collect each 8 serial bits (byte) in one packet. There is a valid signal that became high when the byte is ready to be sent to the PPI.

ESC_DECODER

The Escape Decoder block is responsible for extracting data from the differential lines during escape mode. The escape clock is generated by performing an exclusive OR operation on the input signals: Clock_Esc = Dp ^ Dn.
The data bit is decoded from the differential pair using the logic expression: Data_Coded = Dp & ~Dn.

Inputs:

- Dp, Dn: Differential data lines from the transmitter.

Outputs:

- Data: The decoded data bit.
- Enable: A control signal indicating valid data, allowing the system to process the received bit.

Counter

This block serves as a bit counter that keeps track of the number of data bits received during escape mode. It helps ensure that the correct number of bits is processed before concluding the operation.

CHAPTER 3: VERIFICATION

3.1 INTRODUCTION

In modern mobile systems, MIPI D-PHY is a key physical layer interface used to connect high-speed devices such as image sensors and displays to application processors. As an integral part of MIPI CSI-2 and DSI protocols, the D-PHY provides both high-speed (HS) and low-power (LP) transmission modes, making it highly efficient for data-intensive and energy-sensitive applications.

This graduation project focuses on verifying the complete functionality of a MIPI D-PHY system, which includes both the Transmitter (TX) and the Receiver (RX). A comprehensive verification environment has been developed using Universal Verification Methodology (UVM), with a reference model for both TX and RX built in MATLAB Simulink.

The goal is to ensure that the DUT (Design Under Test), complies with the MIPI D-PHYv1.2 specification in all operational modes. Verification is achieved by comparing DUT behavior against a golden model through simulation using SystemVerilog, DPI-C, and QuestaSim.

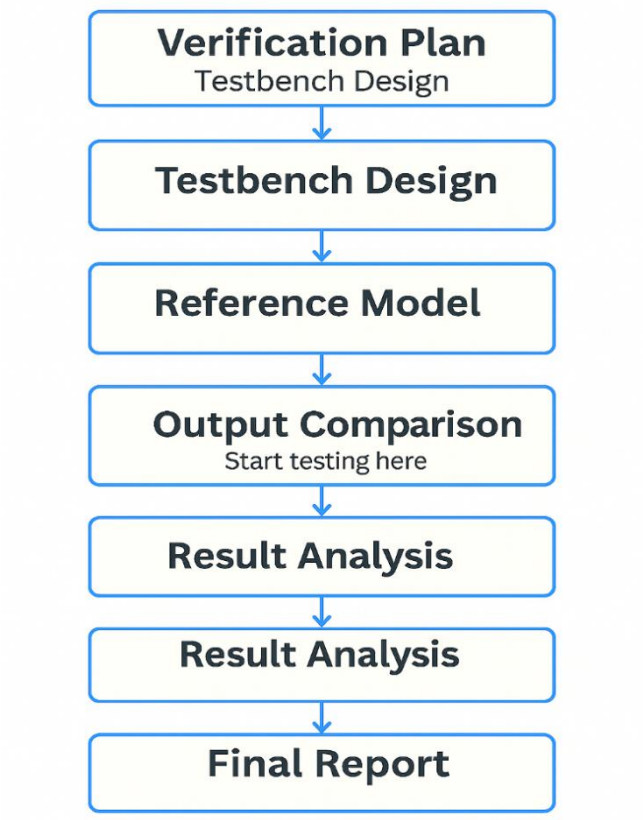


Figure 3.1: Verification flow

3.2 Project Scope and Objectives

functional verification primary objectives are:

- Develop a complete UVM-based testbench to verify TX, RX and FULL DUT.
- Build a behavior Simulink golden reference model for TX, RX.
- Implement DPI-C interfaces to connect SystemVerilog with Simulink models.
- Validate DUT functionality under all MIPI D-PHY operational modes.
- Design a UVM scoreboard to compare RTL and reference outputs in real-time.
- Ensure coverage of protocol features such as HS timing, LP signaling, turnaround behavior, and ULPS entry/exit.

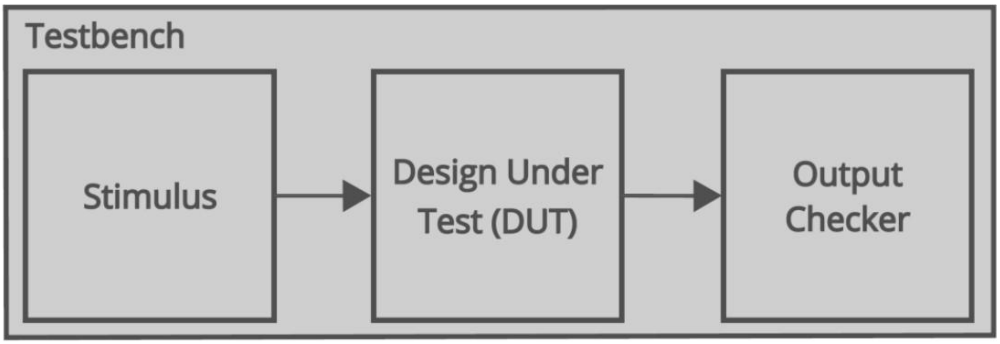


Figure3.2: Test bench architecture

3.3 the test bench architecture

There are two approaches to the test bench architecture:

- **Stand-alone verification phase:** each module—Transmitter (TX) and Receiver (RX)—was independently verified using a golden reference model developed in MATLAB Simulink. The communication between SystemVerilog and Simulink was achieved through a DPI-C interface, allowing synchronized stimulus and output comparison. This approach enabled early detection of design-level issues within each module separately and provided high observability and control.

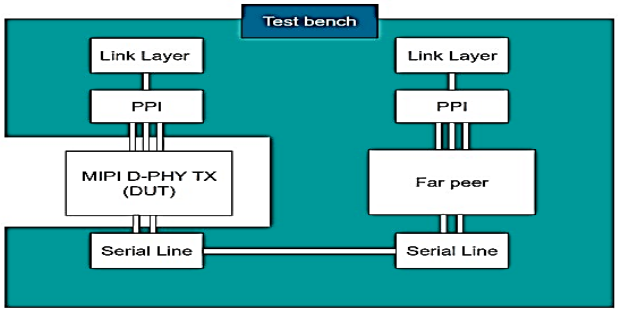


Figure 3.3: Stand-alone testing

- **Back-to-back testing:** employed by directly connecting the TX and RX RTL designs. This setup allowed end-to-end verification of the complete D-PHY system, where the testbench emulated the upper layers (such as PPI) and monitored the correctness of data transfer across the physical layer.

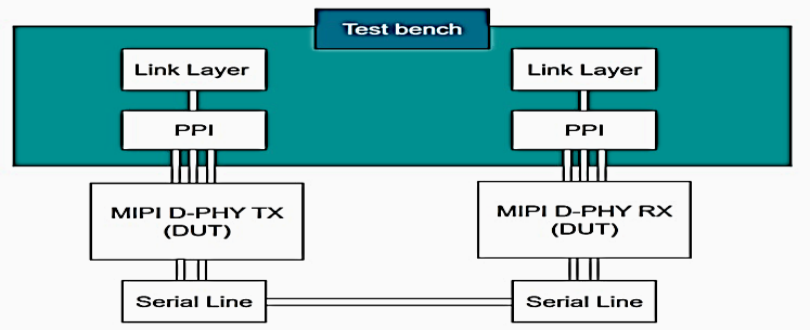


Figure 3.4: Back-to-back testing

This dual-verification strategy allowed cross-validation and ensured that no functional bugs were masked due to mirrored behavior between transmitter and receiver.

3.4 Verification Methodology

The verification environment is structured using Universal Verification Methodology (UVM) for modularity and reusability. The environment includes:

- Interfaces and Transaction Items: Define signal-connections and data structures.
- Agents: Independent agents for TX and RX.
- Driver & Monitor Components: Responsible for applying stimulus and capturing DUT behavior.
- Scoreboard: Performs cycle-accurate and signal-accurate comparisons with reference model.
- Simulink Models: Represent TX, RX, and the full system behavior.
- DPI-C Wrapper: Acts as a communication bridge between SystemVerilog and MATLAB.

Simulation is run using QuestaSim, with synchronization between the SystemVerilog simulator and Simulink runtime achieved using DPI function calls.

3.4.1 UVM-SV

The Universal Verification Methodology (UVM) is a standardized, vendor-independent methodology for verifying complex digital integrated circuits. It is built on SystemVerilog and provides a set of base classes that allow engineers to construct modular, reusable, and scalable verification environments.

UVM is primarily derived from OVM (Open Verification Methodology), which itself was influenced by the eRM (e Reuse Methodology) developed for the e Verification Language by Verisity Design in 2001.

Unlike previous methodologies that were tied to specific simulator vendors, UVM is an Accellera standard, supported by multiple EDA tool providers such as Cadence, Synopsys, Mentor Graphics (Siemens EDA), Aldec, and Xilinx (XSIM).

The UVM class library extends SystemVerilog by offering:

- Automation features like (copy, compare, print, record, pack, and unpack)
- Transaction-level modeling using sequences
- Standard interfaces for building verification components (UVCs/VIPs)
- Built-in support for phase-based simulation control (build, connect, run, etc.)

By using UVM, teams can accelerate development, enhance code reuse across projects, and improve verification quality. It is now the de facto industry standard for functional verification of ASIC and FPGA designs.

3.4.2 UVM Class Library and Component Architecture

The UVM class library provides a comprehensive set of utilities that support the construction of modular and reusable verification environments. It introduces multiple abstraction layers and encapsulates best practices for developing testbenches for complex digital designs.

Key features of the UVM library include:

- Configuration Database for centralized setup and parameter passing
- Transaction-Level Modeling (TLM) for communication between components
- Data Automation Methods: such as copy, print, compare, pack, and unpack
- Hierarchical component structure that mirrors the design architecture

3.4.3 UVM Component Hierarchy Environment:

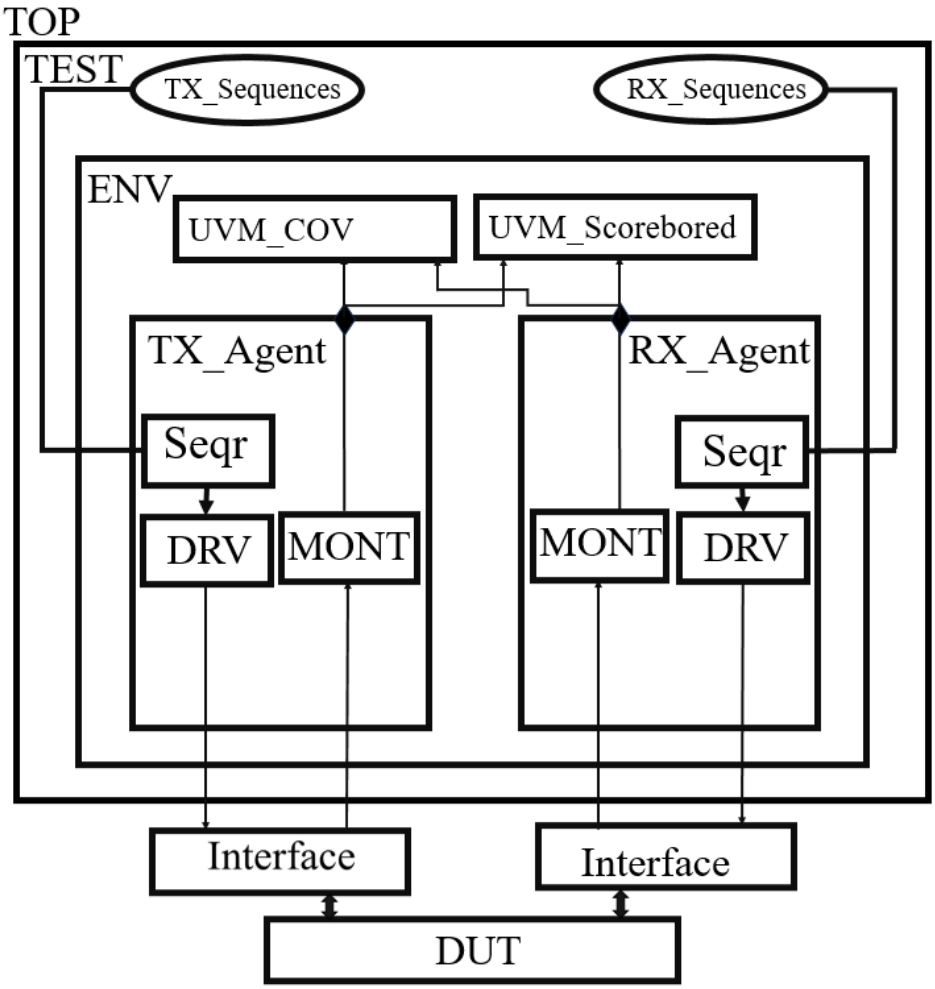


Figure 3.5: UVM ENV

- **TOP**: Instantiates both the DUT and the Test class.
- **Test**: A container that controls which environment and sequences are executed during simulation.
- **Environment (env)**: Contains all verification agents, the scoreboard, and functional coverage components.
- **Agent**: May be active or passive:
    - **Active Agent** includes:
        - **Sequencer**: Drives sequence items to the driver
        - **Driver**: Applies stimulus to the DUT interface
        - **Monitor**: Observes DUT outputs for checking or coverage

- **Passive Agent** includes:
  - **Monitor** only (no driver or sequencer)

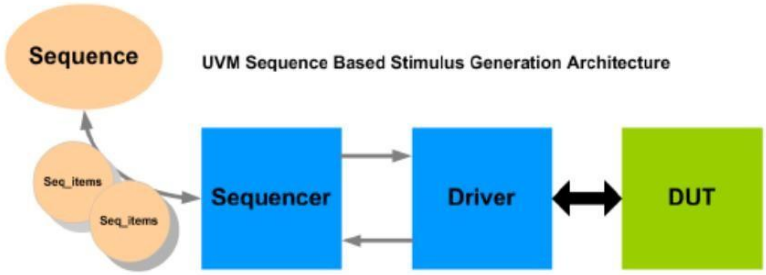


Figure 3.6: UVM Sequence Based Stimulus Generation Architecture

- **Coverage Collector:** Measure functional completeness of tests.
- **Scoreboard**: Performs data checking by comparing DUT outputs against expected values, ensuring data integrity.
- **Interface**: Connects DUT signals to the testbench using a well-defined signal grouping mechanism.
- **Sequence Item (uvm_sequence_item)**: Represents a transaction; contains randomized fields (e.g., data, control bits).
- **Sequence (uvm_sequence)**: Generates randomized sequence items and controls their order and content.
- **Virtual Sequence**: A higher-level sequence that coordinates multiple sequences, often across multiple agents.
- **Sequencer**: Sends sequence items to the driver based on sequence behavior.
- **Driver**: Converts sequence items into actual pin-level signals and applies them to the DUT.

3.4.4 The Typical UVM Testbench Architecture

The Universal Verification Methodology (UVM) defines a highly modular and layered architecture for building reusable and scalable testbenches. It leverages the SystemVerilog class library, which provides generic utilities such as component hierarchies, a transaction-level modeling (TLM) framework, configuration databases, and factory mechanisms to instantiate and configure verification components dynamically.

While various architectures can be implemented using UVM, this section presents the most typical and recommended structure to introduce key components and terminology.

• UVM Testbench

The UVM Testbench typically instantiates the Design under Test (DUT) module and the UVM Test class and configures the connections between them. If the verification collaterals include module-based components, they are instantiated under the UVM Testbench as well. The UVM Test is dynamically instantiated at run-time, allowing the UVM Testbench to be compiled once and run with many different tests. Note that in some architectures, the Testbench term is used to refer to a special module that encapsulates verification collaterals only, which in turn are integrated up with the DUT.

• UVM Test

The UVM Test is the top-level UVM Component in the UVM Testbench. The UVM Test typically performs three main functions: Instantiates the top-level environment, configures the environment (via factory overrides or the configuration database), and applies stimulus by invoking UVM Sequences through the environment to the DUT. Typically, there is one base UVM Test with the UVM Environment instantiation and other common items. Then, other individual tests will extend this base test and configure the environment differently or select different sequences to run.

• UVM Environment

The UVM Environment is a hierarchical component that groups together other verification components that are interrelated. Typical components that are usually instantiated inside the UVM Environment are UVM Agents, UVM Scoreboards, or even other UVM Environments. The top level UVM Environment encapsulates all the verification components targeting the DUT. For example: In a typical system on a chip (SoC) UVM Environment, you will find one UVM Environment per IP (e.g., PCIe Environment, USB Environment, Memory Controller Environment, etc.). Sometimes, those IP Environments are grouped together into Cluster Environments (e.g., IO Environment, Processor Environment, etc.) that are grouped together eventually in the to- level SoC Environment.

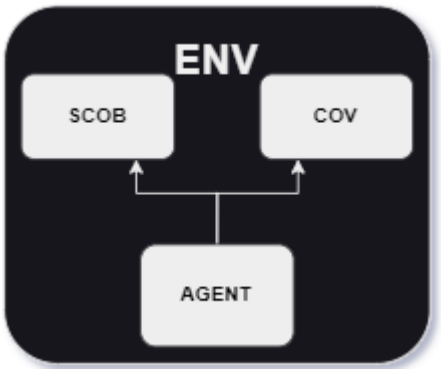


Figure 3.7: UVM ENV Component

• UVM Scoreboard

The UVM Scoreboard's main function is to check the behavior of a certain DUT. The UVM Scoreboard usually receives transactions carrying inputs and outputs of the DUT through UVM Agent analysis ports, runs the input transactions through some kind of a reference model (also known as the predictor) to produce expected transactions, and then compares the expected output versus the actual output. There are different methodologies on how to implement the scoreboard, the nature of the reference model, and how to communicate between the scoreboard and the rest of the testbench.

- **Coverage Collector**

Coverage as applied to hardware verification languages refers to the collection of statistics based on sampling events within the simulation. Coverage is used to determine when the device under test (DUT) has been exposed to a sufficient variety of stimuli that there is a high confidence that the DUT is functioning correctly. Note that this differs from code coverage which instruments the design code to ensure that all lines of code in the design have been executed. Functional coverage ensures that all desired corner and edge cases in the design space have been explored. A SystemVerilog coverage group creates a database of "bins" that store a histogram of values of an associated variable. Cross-coverage can also be defined, which creates a histogram representing the Cartesian product of multiple variables.

• UVM Agent

The UVM Agent is a hierarchical component that groups together other verification components that are dealing with a specific DUT interface. A typical UVM Agent includes a UVM Sequencer to manage stimulus flow, a UVM Driver to apply stimulus on the DUT interface, and a UVM Monitor to monitor the DUT interface. UVM Agents might include other components, like coverage collectors, protocol checkers,

a TLM model, etc. 61 Figure 23: UVM Agent The UVM Agent needs to operate both in an active mode (where it is capable of generating stimulus) and a passive mode (where it only monitors the interface without controlling it).

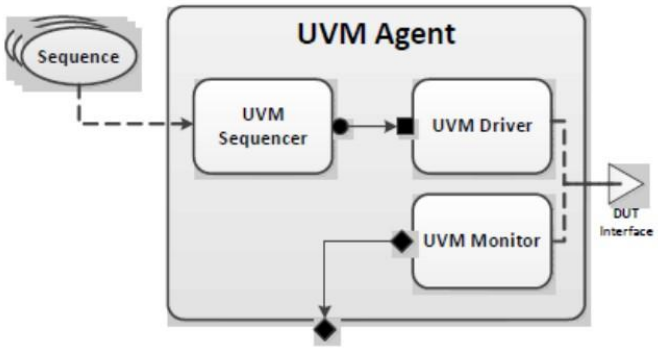


Figure 3.8: UVM AGENT

• UVM Sequencer

The UVM Sequencer serves as an arbiter for controlling transaction flow from multiple stimulus sequences. More specifically, the UVM Sequencer controls the flow of UVM Sequence Items transactions generated by one or more UVM Sequences.

• UVM Sequence

A UVM Sequence is an object that contains a behavior for generating stimulus. UVM Sequences are not part of the component hierarchy. UVM Sequences can be transient or persistent. A UVM Sequence instance can come into existence for a single transaction, it may drive stimulus for the duration of the simulation, or anywhere in-between. UVM Sequences can operate hierarchically with one sequence, called a parent sequence, invoking another sequence, called a child sequence. To operate, each UVM Sequence is eventually bound to a UVM Sequencer. Multiple UVM Sequence instances can be bound to the same UVM Sequencer.

• UVM Driver

The UVM Driver receives individual UVM Sequence Item transactions from the UVM Sequencer and applies (drives) it on the DUT Interface. Thus, a UVM Driver spans abstraction levels by converting transactionlevel stimulus into pin-level stimulus. It also has a TLM port to receive transactions from the Sequencer and access to the DUT interface to drive the signals.

• **UVM Monitor**

The UVM Monitor samples the DUT interface and captures the information there in transactions that are sent out to the rest of the UVM Testbench for further analysis. Thus, similar to the UVM Driver, it spans abstraction levels by converting pin-level activity to transactions. In order to achieve that, the UVM Monitor typically has access to the DUT interface and also has a TLM analysis port to broadcast the created transactions through. The UVM Monitor can perform internally some processing on the transactions produced (such as coverage collection, checking, logging, recording, etc.) or can delegate that to dedicated components connected to the monitor's analysis port.

- **UVM Sequence Item**

The uvm_sequence_item is the fundamental unit of stimulus in the UVM methodology. It represents a transaction, which is an abstract data structure that encapsulates the necessary information to drive the DUT or to analyze its behavior.

Sequence items are typically used by sequences to describe:

- Data Randomization using rand and constraints
- Control signals Automation methods like copy(), compare(), print()
- Packet fields Field macros (uvm_field_*) for enabling automation and introspection

- **Constrained Random Generation**

Integer quantities defined either in a class definition or as stand-alone variables in some lexical scope, can be assigned random values based on a set of constraints. This feature is useful for creating randomized scenarios for verification. Within class definitions, the rand and randc modifiers signal variables that are to undergo randomization. randc specifies permutation-based randomization, where a variable will take on all possible values once any value is repeated. Variables without modifiers are not randomized.

- **The role of "Constraints"**

- More focused on the interesting conditions.

- To avoid illegal/uninteresting conditions.

- Steer random choices with constraints and weighted probabilities.

- Target rare or low-likelihood conditions by assigning higher weight.

- **Controlling constraint**

The constraint_mode() and the random_mode() methods are used to control the randomization. constraint_mode() is used to turn a specific constraint on and off and the random_mode is used to turn a randomization of a specific variable on or off.

Constrained parameters and unconstrained parameters

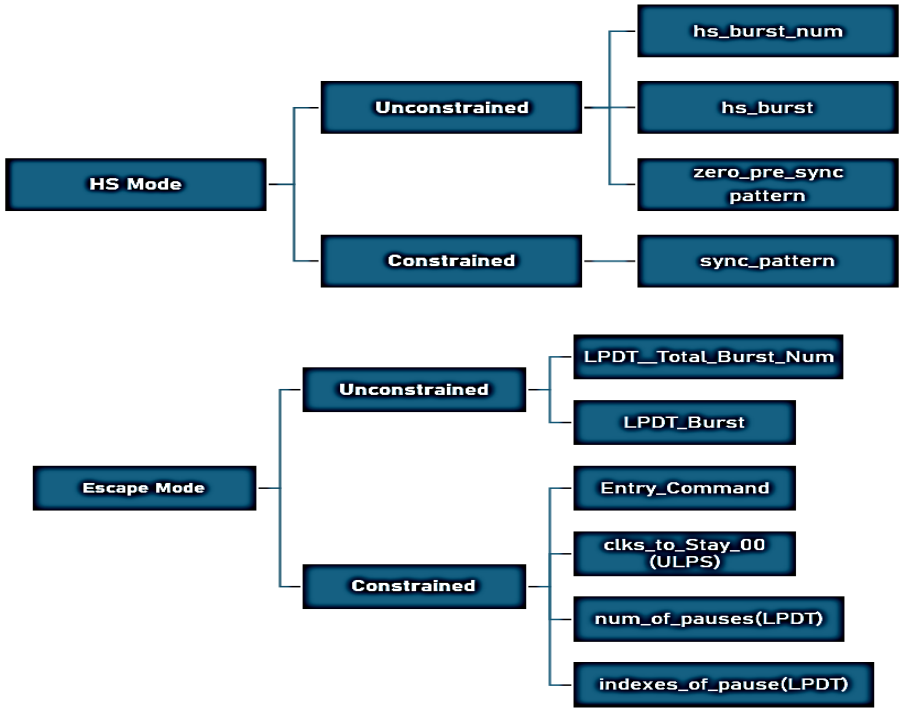


Figure 3.9: Constrained parameters and unconstrained parameters

- **Assertion**

Assertions are used to validate design whether it is working correctly or not. Assertions can be useful to make sure 'How good is the test case?' Furthermore, it provides a means to measure the quality of the verification process through the creation of coverage using cover property feature of System Verilog assertion. An assertion adds an advantage in debugging process and makes complex simulation debug easy.

• In System Verilog there are two types of assertions:

1. Immediate Assertions Immediate Assertion as name implies, execute immediately, in zero simulation time. It can be used in initial and always procedures, task and functions. This type of assertion performs a true/false. If the test result is true, it executes the pass statement and if the test result is false or unknown, it executes false statement. Pass statements are optional; most engineers don't prefer to specify pass statements as they are more interested on failures.

2. Concurrent Assertions A concurrent assertion uses a clock to trigger the assertions evaluation. The difference between concurrent and immediate assertions is that concurrent assertions evaluate condition over time, whereas immediate assertions test at the point in time when the assertion is called. These assertions can be specified in initial block, always block or standalone like continues assignment.

• **Advantages of Assertions:**

1. Dozens of lines of code can be represented in one line using SVA code.
2. Assertions can be controlled and can be disabled at any point during the simulations. SVA can be turned off during the reset or until the simulation reaches the particular event or logic.
3. Assertion can have severity levels; failures can be non-fatal or fatal errors.
4. SVA can be ignored by synthesis; designer does not need to include translate_on and transalate_off throughout the RTL code.
5. Assertions can be used in simulation and formal verification.

3.4.5 UVM Hierarchy

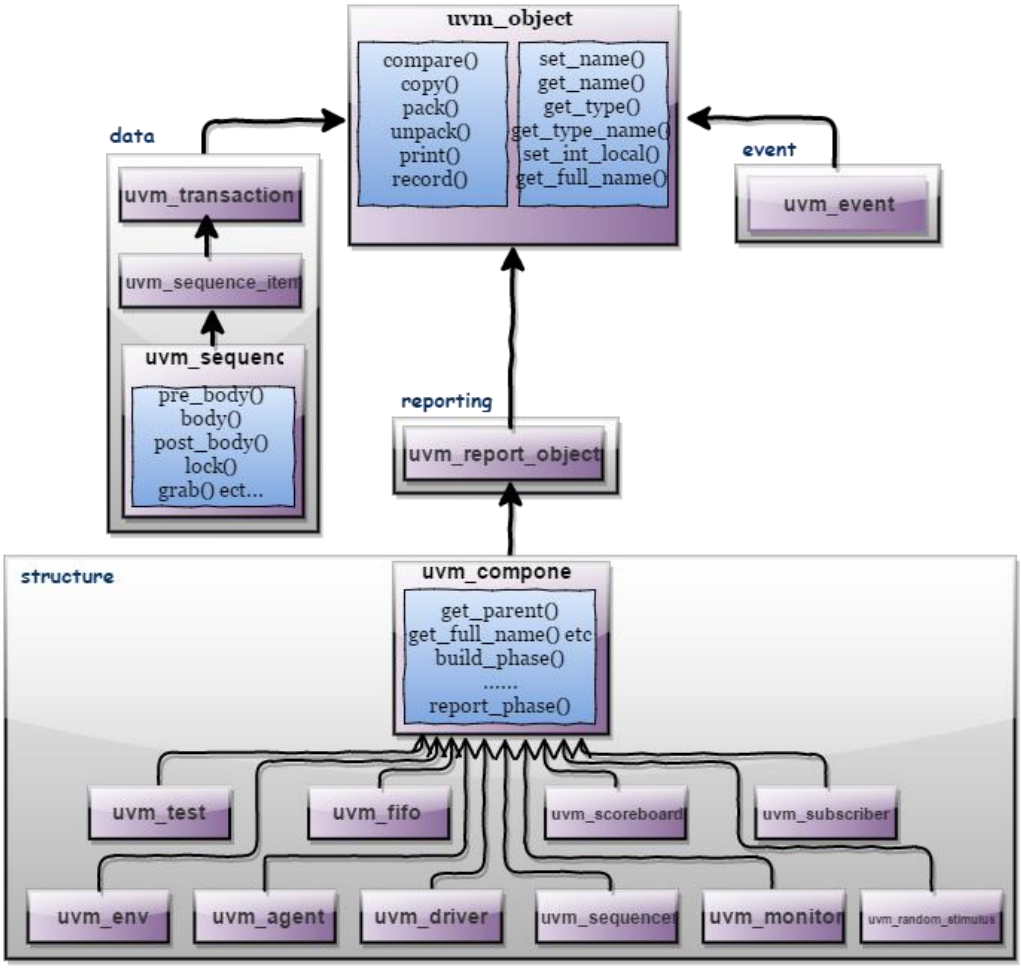


Figure 3.10: UVM hierarchy

The uvm_object class is the base class for all UVM data and hierarchical classes. Its primary role is to define a set of methods for such common operations as create, copy, compare, print, and record. uvm_objects are used for UVM transient objects like uvm_seq_item, uvm_transaction, uvm_sequence. The base class uvm_object is also used for configuration objects, i.e. classes which contain configuration data to configure uvm_components or other uvm_objects, like sequences, etc. uvm_components are "static-like" in that they are created during build_phase() and persist throughout the simulation. Think of them as the class-based equivalent of modules. uvm_objects are transient, such as transactions that are created when needed and disappear when not used anymore.

- UVM phases

In order to have a consistent testbench execution flow, the UVM uses phases to order the major steps that take place during simulation. There are three groups of phases, which are executed in the following order:

1. Build phases - where the testbench is configured and constructed.
2. Run-time phases - where time is consumed in running the testcase on the testbench.
3. Clean up phases - where the results of the testcase are collected and reported.

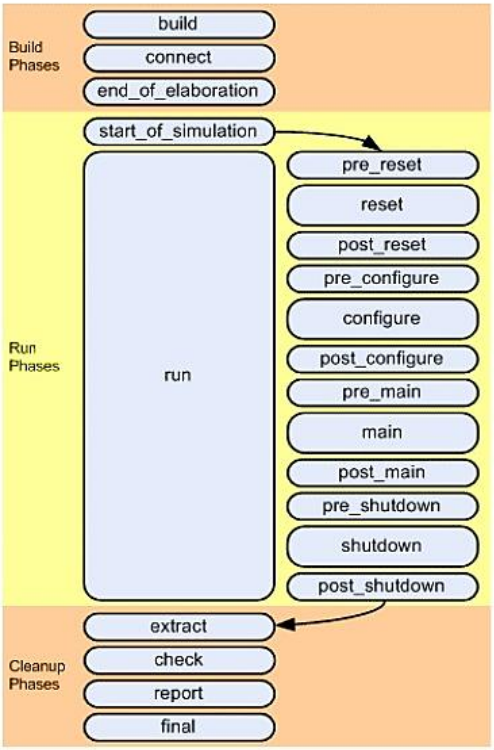


Figure 3.11: UVM phases hierarchy

3.4.6 UVM Dynamic objects Sequence

UVM sequences are made up of several data items which can be put together in different ways to create interesting scenarios. They are executed by an assigned sequencer which then sends data items to the driver. Sequences are the primary means of generating stimulus in the UVM. Sequences bring an Object Orientated approach to stimulus generation that is very flexible and provides new options for generating stimulus. An UVM sequence contains a task called body. When a sequence is used, it is created, then the body method is executed, and then the sequence can be discarded. Unlike an uvm_component, a sequence has a limited simulation lifetime and can therefore be described as a transient object. The sequence body method can be used to generate sequence_item objects which are sent to a driver component, via a sequencer component, for conversion into pin-level activity or it can be used to do some combination of the two. When a response is generated by the DUT, then a sequence_item is used by the driver to pass the response information back to the

originating sequence, again via the sequencer. In both environments we have sequences for each mode of operation.

3.4.6.1 List of RX Verification Sequences

- sequence_HS_RX_8_10
  - Description: Stimulates the RX interface with 8-bit payloads over 10 high-speed cycles.
  - Purpose: Validate correct reception and alignment of HS-mode data on the RX side.
- sequence_ESC_LP_RX_8_50
  - Description: Sends 8-bit payloads over 50 escape-mode cycles, using one-hot space encoding.
  - Purpose: Verify RX support for low-power data transmission in escape mode and correct decoding of one-hot symbols.
- sequence_ESC_ULP_RX_50
  - Description: Drives the RX into Ultra Low Power State (ULPS) for 50 cycles.
  - Purpose: Confirm correct RX behavior during ULPS entry, retention, and exit sequences.
- sequence_ESC_TRIGGER_RX
  - Description: Sends a sequence of 4 escape-mode triggers, using one-hot encoding.
  - Purpose: Validate trigger recognition and proper decoding by the RX logic.
- sequence_HS_ERR_CHECK_RX
  - Description: Applies error-prone HS-mode inputs and monitors the RX response.
  - Purpose: Verify that RX asserts appropriate error flags (e.g., ErrSyncEsc, ErrContentionEsc) under abnormal HS conditions.
- sequence_ESC_ERR_CHECK_RX
  - Description: Applies invalid or unexpected escape-mode conditions.
  - Purpose: Validate RX response to escape-mode protocol violations and ensure assertion of ESC-mode error signals.

3.4.6.2 List of TX Verification Sequences

- Reset_sequence
  - Description: Deriving the stimulus as following all=0 => enables=1 => RST=1 => enables=0 => 1000 repetitions with RST=1, randomize all => 1000 repetitions with RST=0, randomize all.
  - Purpose: Covering all cases when resetting the system.

- HS_sequence
  - Description: Deriving the stimulus as following TxRequestHS=1, RST=1, enables=1, the rest=0 => TxDataHS=6 => 1000 repetitions with randomizing TxDataHS => 1000 repetitions with randomizing all except TxRequestHS=1, RST=1, enables=1.
  - Purpose: Covering all cases of High-Spead mode.
- Lpdts_sequence
  - Description: Deriving the stimulus as following TxRequestEsc=1, RST=1, enables=1, TxLpdtEsc=1, the rest=0 => TxDataEsc=4, TxValidEsc=1 => 1000 repetitions with randomizing TxDataEsc => 1000 repetitions with TxLpdtEsc=0, randomizing TxDataEsc => 1000 repetitions with randomizing all except TxLpdtEsc=1, RST=1, enables=1, TxValidEsc=1 => 1000 repetitions with randomizing all except TxLpdtEsc=1, RST=1, enables=1.
  - Purpose: Covering all cases of Low_power_data_transimitter_state mode.
- Trigger_sequence
  - Description: Deriving the stimulus as following TxRequestEsc=1, RST=1, enables=1, the rest=0 => TxTriggerEsc=0001 => TxRequestEsc=0, TxTriggerEsc=0 => TxRequestEsc=1, TxTriggerEsc=0010 => TxRequestEsc=0, TxTriggerEsc=0 => TxRequestEsc=1, TxTriggerEsc=0100 => TxRequestEsc=0, TxTriggerEsc=0 => TxRequestEsc=1, TxTriggerEsc=1000 => TxRequestEsc=0, TxTriggerEsc=0 => 1000 repetitions with randomizing TxTriggerEsc => TxRequestEsc=0, TxTriggerEsc=0 => 1000 repetitions with randomizing all except TxRequestEsc=1, TxTriggerEsc =1, RST=1, enables=1.
  - Purpose: Covering all cases of Trigger mode.
- Ulps_sequence
  - Description: Deriving the stimulus as following TxRequestEsc=1, RST=1, enables=1, TxUlpsClk=1, TxUlpsEsc=1, the rest=0 => TxUlpsEsc=0 => 1000 repetitions with randomizing all except TxRequestEsc=1, TxUlpsEsc=1, TxUlpsClk=1, RST=1, enables=1 => TxRequestEsc=1, RST=1, enables=1, TxUlpsClk=1, TxUlpsEsc=1, the rest=0 => TxUlpsEsc=0, TxRequestEsc=0, TxUlpsClk=1, CLK_TxULPSExit=1, DATA_TxUlpsExit=1.
  - Purpose: Covering all cases of Ultra_low_power_state mode.
- Main_sequence
  - Description: 1000 repetitions with randomizing all signals.
  - Purpose: Covering as many states as possible.

3.4.6.3 List of FULL SYS Verification Sequences

- sequence_HS_FULL_8_1000
  - Description: Stimulates interface with 8-bit payloads over 1000 high-speed cycles.
  - Purpose: Validate correct reception and alignment of HS-mode data.
- sequence_ESC_LP_FULL_8_10
  - Description: Sends 8-bit payloads over 50 escape-mode cycles, using one-hot space encoding.
  - Purpose: Verify low-power data transmission in escape mode and correct decoding of one-hot symbols.
- sequence_ESC_ULP_FULL_50
  - Description: Drives the Ultra Low Power State (ULPS) for 50 cycles.
  - Purpose: Confirm correct behavior during ULPS entry, retention, and exit sequences.
- sequence_ESC_TRIGGER_FULL
  - Description: Sends a sequence of 4 escape-mode triggers, using one-hot encoding.
  - Purpose: Validate trigger recognition and proper decoding.
- sequence_HS_ERR_CHECK_FULL
  - Description: Applies error-prone HS-mode inputs and monitors response.
  - Purpose: Verify asserts appropriate error flags (e.g., ErrSyncEsc, ErrContentionEsc) under abnormal HS conditions.
- sequence_ESC_ERR_CHECK_ FULL
  - Description: Applies invalid or unexpected escape-mode conditions.
  - Purpose: Validate response to escape-mode protocol violations and ensure assertion of ESC-mode error signals.

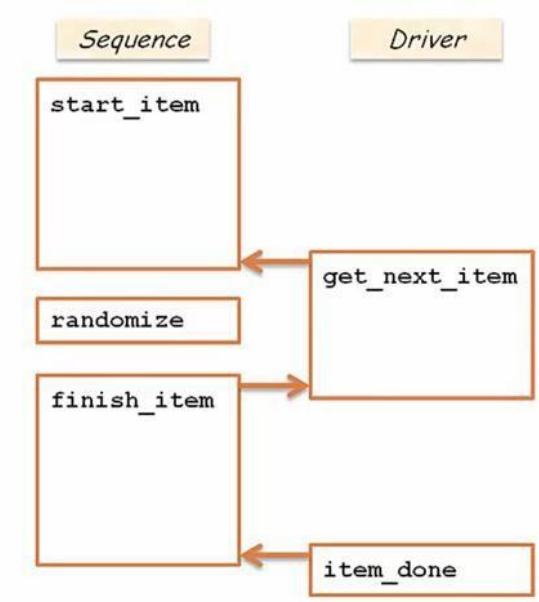


Figure 3.12: Describes the handshake between the driver and the sequence

3.4.6.4 Virtual Sequence

A virtual sequence is a sequence which controls stimulus generation using several sequencers. Since sequences, sequencers, and driver are focused on point interfaces, almost all testbenches require a virtual sequence to coordinate the stimulus across different interfaces and the interactions between them. A virtual sequence is often the top level of the sequence hierarchy. A virtual sequence differs from a normal sequence in that its primary purpose is not to send sequence items. Instead, it generates and executes sequences on different target agents. To do this it contains handles for the target sequencers, and these are used when the sequences are started.

In UVM: we decide to not have the virtual sequence and run test scenarios in the test class itself.

3.4.6.5 Configuration

DB The uvm_config_db is a UVM utility class that is used to pass configuration data objects between component objects in a UVM testbench. Any call to the uvm_config_db is parameterised with the type of data object that is stored or retrieved. To store an object in the uvm_config_db, its set () method is used, and to retrieve an object from the uvm_config_db, it get () method is used. Both these methods have arguments that assign the object, associate the object with a lookup key; and define which components within the UVM testbench hierarchy can reference the object.

In SVUVM: we use this DB to set the interface's instances in the top module and get them in all components need them.

3.5 DPI-C Integration Between MATLAB Simulink and UVM

3.5.1 Introduction: Co-Simulation Between MATLAB and SystemVerilog

Modern SoC designs and PHY layer protocols require thorough and reliable functional verification. To meet these needs, we implemented a live and fully synchronized co-simulation between the RTL DUT and a high-level behavioral model created in MATLAB Simulink. The methodology uses the Direct Programming Interface for C (DPI-C), which enables the UVM testbench in SystemVerilog to interact with the compiled model (DLL) generated by MATLAB. This integration serves as a golden reference and provides a mechanism to validate the functional correctness of the RTL

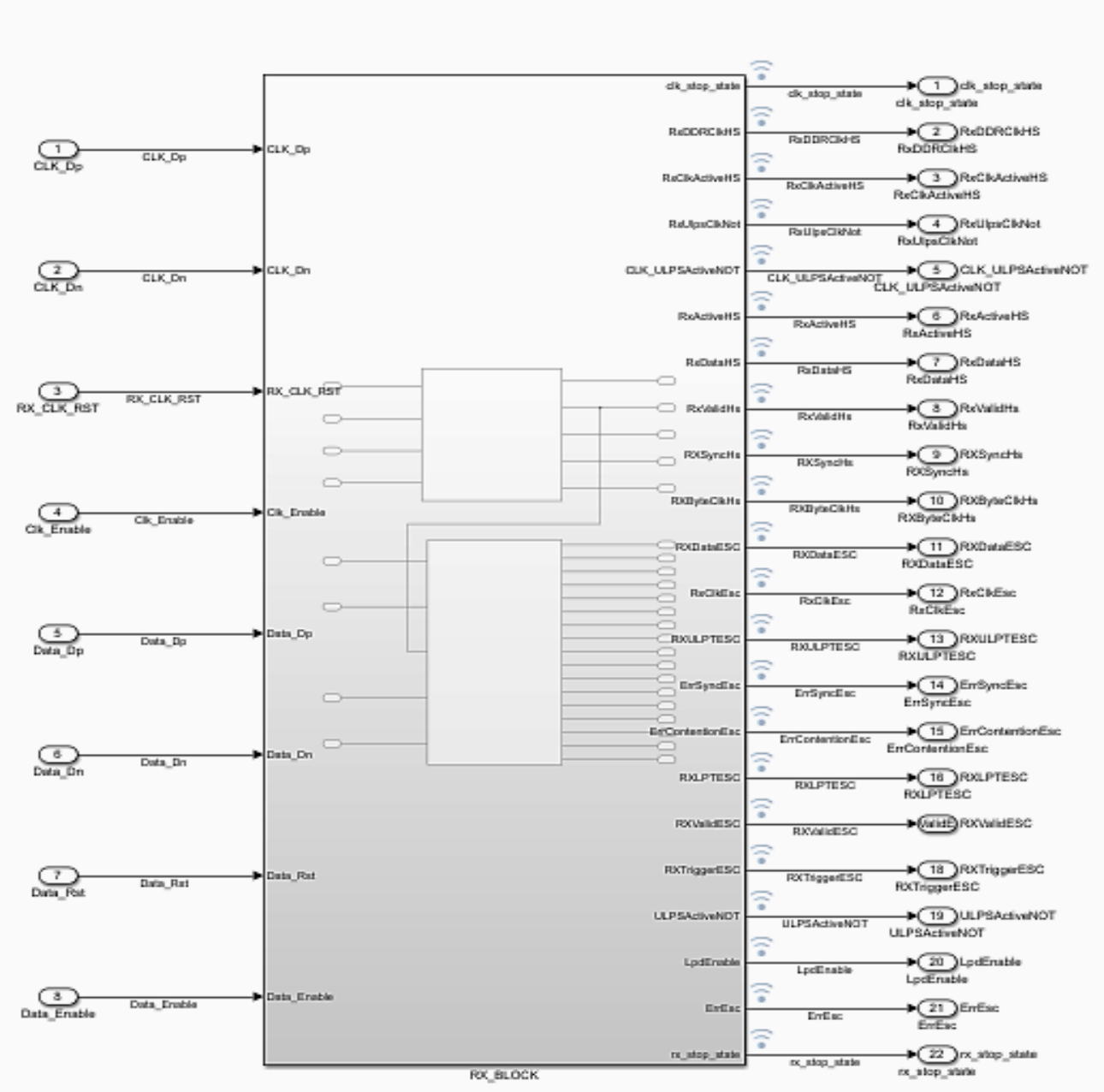


Figure 3.13: MATLAB RX MODELING

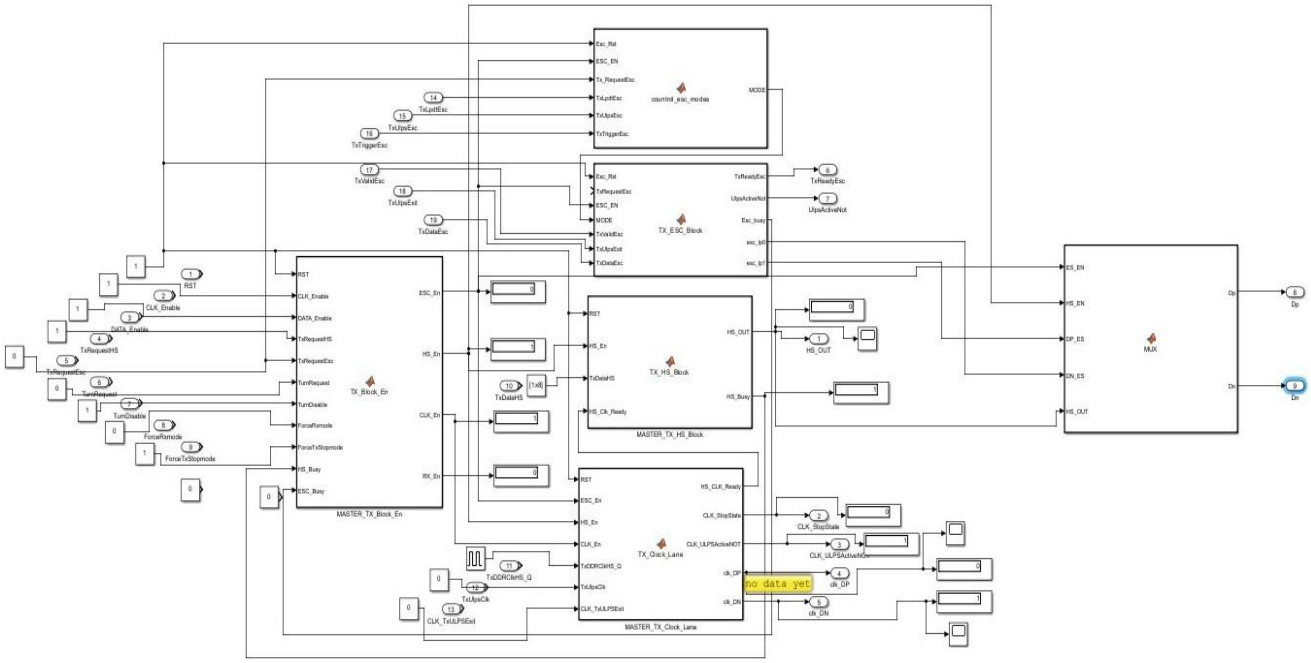


Figure 3.14: MATLAB TX MODELING

3.5.2 DPI-C Technology Overview

The Direct Programming Interface for C (DPI-C) is defined by the IEEE 1800 SystemVerilog standard. It facilitates interaction between SystemVerilog and C-language functions. SystemVerilog code can call C functions, and C code can also call exported SystemVerilog functions. DPI-C allows the simulation engine to interface with C-based models in a native and efficient manner, without overhead of socket communication or external scripting.

For our project, DPI-C was used to call compiled MATLAB models that reflect the expected behavior of MIPI D-PHY RX modules. These models were exported from Simulink as DLLs and wrapped in a C interface to make them DPI-compatible.

3.5.3 Integration Workflow in Detail

The integration process spans both MATLAB and SystemVerilog environments. The workflow is broken into five main phases:

1. **Modeling Phase (MATLAB/Simulink)**
    - A Simulink model is constructed to represent the behavioral logic of the D-PHY RX. It covers different modes (HS, LPDT, ULPS, etc.) and captures protocol-specific timing and data interpretation.

- The model is modular and simulation-ready, using blocks and functions native to Simulink to enable time-step execution.

2. **Code Generation Phase (Simulink Coder)**
    - Using Simulink Coder, the behavioral model is transformed into C code. This includes all computational blocks and control logic.
    - The generated code includes standard APIs such as initialize(), output(), and update () functions.
3. **Compilation Phase (DLL Generation)**
    - The generated C code is compiled using a supported C compiler (e.g., Microsoft Visual Studio) into a Windows Dynamic Link Library (DLL).
    - The resulting DLL includes all logic required to simulate the RX model from an external simulator.
4. **Wrapper Development (C DPI Layer)**
    - A wrapper C file is written to expose a DPI-compliant interface. Functions such as set_input_xxx(double val), run_step(), and get_output_xxx() are defined.
    - These functions act as the communication bridge between UVM and MATLAB logic.
5. **SystemVerilog DPI Package (UVM Side)**
    - A SystemVerilog package is created to import all C functions using DPI.
    - The DPI functions are used in UVM components like drivers and scoreboards to exchange data and control simulation flow.

3.5.4 Detailed Functional Interaction

At simulation runtime, a tightly synchronized communication protocol is established between the UVM testbench and the MATLAB-based model via the DPI wrapper. The cycle-by-cycle operations include:

1. **Input Transfer:** UVM sends values to the wrapper using DPI setter functions.
2. **Simulation Step:** A run_step() function is invoked, which updates the MATLAB model for one simulation tick.
3. **Output Retrieval:** The results from the Simulink model are accessed via getter functions.

4. **Comparison:** The UVM scoreboard evaluates the RTL outputs against the reference model outputs and flags mismatches.

This interaction guarantees that the RTL DUT and the Simulink model are stimulated with the same data and observed at the same simulation instance.

3.5.5 DPI-C Wrapper Design

The DPI wrapper (dpi_wrapper.c) is the core of the interface. It consists of the following parts:

- Global variables used to store input/output values.
- Setter functions (e.g., void set_data_dp(double val);) that update inputs before simulation step.
- void run_step(); that internally calls MATLAB model step functions (e.g., RX_model_step();).
- Getter functions (e.g., double get_rx_data();) to retrieve results post-simulation.

Each C function is declared in a header file (dpi_header.h) and properly exposed to SystemVerilog using extern "C" for compatibility.

3.5.6 UVM Side Usage

In the UVM environment:

- DPI imports are declared in dpi_pkg.sv using import "DPI-C" function ... syntax.
- The driver or monitor components call setter functions at each cycle.
- The scoreboard coordinates calls to run_step() and collects outputs.

This design ensures modularity, allowing the DPI logic to be updated independently from the rest of the testbench.

3.5.7 Advantages and Verification Benefits

The integration yielded numerous benefits:

- **Real-Time Co-Simulation:** MATLAB model reacts to same data as DUT and produces synchronized outputs.
- **Formal Comparison:** Functional discrepancies are immediately flagged with full signal-level traceability.
- **Reference Golden Model:** MATLAB model acts as a trusted source during protocol verification.

- **Better Debugging:** Mismatches help locate bugs in either the RTL or testbench logic.
- **Scalability:** Future changes to the model or DUT can be retested easily.

3.5.8 Performance and Limitations

To optimize performance and reliability:

- All signals are treated with consistent double datatype.
- Simulation step length is kept minimal to reduce co-simulation latency.
- DPI call overhead is minimized by grouping signal sets.
- Care is taken to synchronize timing across the model and RTL clock edges.

Some limitations observed include:

- Increased simulation runtime due to DLL interaction.
- Complexity in handling signal mapping during debugging.

3.5.9 Future Extensions

This DPI-C methodology opens the door to several future enhancements:

- Integration with more complex models (e.g., multichannel RX)
- Support for bidirectional data flow (TX ↔ RX)
- Coverage-driven stimulus feedback from MATLAB to UVM

Such extensions would increase the realism and coverage of the test environment.

3.5.10 Conclusion

Integrating MATLAB Simulink models into the SystemVerilog UVM environment using DPI-C provides a powerful and scalable verification solution. It bridges the abstraction gap between algorithm-level modeling and RTL design, enabling real-time verification of functional correctness. This approach is not only aligned with industry best practices, but also provides a flexible platform for continuous integration and regression testing in high-speed serial protocols like MIPI D-PHY.

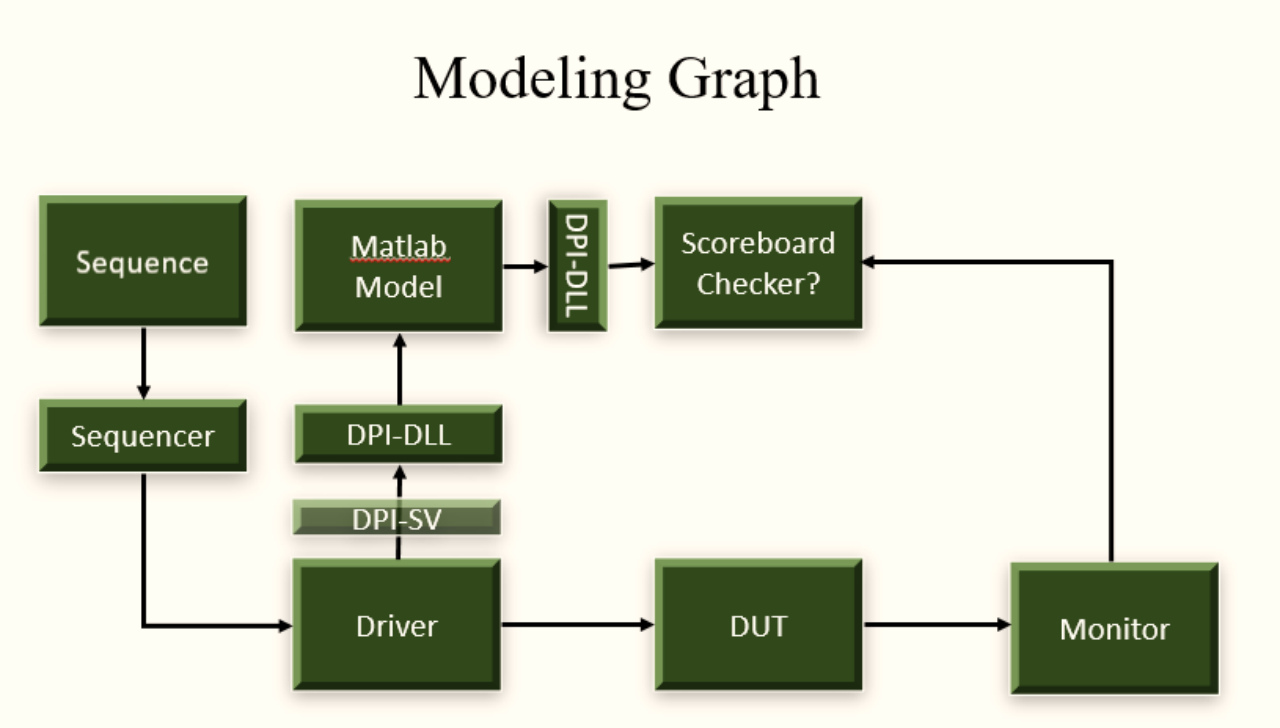


Figure 3.15: MATLAB INTEGRATION WITH UVM ENV

"The DUT design was implemented in MATLAB Simulink and exported to C-code using Simulink Coder. The UVM testbench interacts with it via DPI-C, treating it as a cycle-accurate golden reference model for comparison against the RTL implementation."

3.6 Verification Plans

3.6.1Verification Plan RX

VP_Slave Table from RX_MIPI D-PHYScope	**Design Requirement Description**	**Stimulus Generation**	**Functionality Check**
RX-1	Intialize the system sends	send DP = 1 & DN = 1 & force_rx_mode =1 for 100us	Stopstate=1 &&UlpsActiveNot=1 && RxUlpsClkNot=1 && RxClkActiveHS=0 && RxDDRClkHS=0 &&direction =0

RX-2	HS Mode CLK Lane	CLK_dp & CLK_dn {HS-0 >> LP-00 >> HS-0 >> drive HS clk 200MHZ}	Observe DDR clk & Byte clk & Active hs clk= 1
RX-3	HS Mode Data Lane	DP & DN {HS-0 >> LP-00 >> HS-0 >> HS-seq 00011101}	Observe rx_sync_hs = 1 for one byte clk cycle & rx_active_hs = 1
RX-4	Send 250 byte of data in HS mode	Randomize under constrain 10% max,10%low,80% all value	Stopstate=0 && RxClkActiveHS=1 && RxDDRClkHS= input ddr clk && RXActiveHs = 1 && RxValidHs = 1 && RxDataHS = Data in && SOT_Sync_Error = 0
RX-5	Exite HS mode Data Lane	send the negative last bit in hs data for Tclk_trail about 4 byte >> LP11	Stopstate_data=1 && UlpsActiveNot=1 && RxUlpsClkNot=1 && RXActiveHs = 0 && RxValidHs = 0
RX-6	EXIT HS mode Clk Lane	after detecting lp11 on dp dn send ddrclk for Tcktrail >> LP1 for Ths_exit	Stopstate_CLK=1 && UlpsActiveNot=1 && RxUlpsClkNot=1 && RxClkActiveHS_clk= 0 && RxDDRClkHS= 0
RX-7	ESC MODE Data Lane	DP & DN {LP10 >> LP-00 >> LP01 >> LP00} all stats forced for Tlpx	Stopstate=0 && Activate the ESC Block and wating for the CMD to detect what mode will validate
RX-8	ESC MODE Data Lane LPDT	Send CMD {11100001}	Stopstate=0 && rx_lpdt_esc = 1
RX-9	Drive 10 byte of date	Send data in one hot space encoding	Stopstate=0 && rx_lpdt_esc = 1 & rx_data_esc = Dtat in DP and DN

RX-10	Exite ESC LPDT	Dp & DN Transition to LP11	Stopstate=1 && rx_lpdt_esc = 0
RX-11	ESC MODE Data Lane ULPDT	Send CMD {00011110} >> CLK lane and Data Lane gose to >>LP10 >> LP00	Stopstate=0 && rx_Ulpdt_esc = 1 && UlpsActiveNot=0 && RxUlpsClkNot= 0 && RxClkActiveHS=0
RX-12	Exite ESC ULPDT	Data lane and CLK lane have to gose to LP10 >> LP11	Stopstate=1 && UlpsActiveNot=1 && RxUlpsClkNot=1 && rx_ulp_esc = 0
RX-13	ESC MODE Data Lane Trigger	Send CMD {b01100010}	Stopstate=0 && rx_lpdt_esc = 0 && rx_ulps_esc = 0
RX-14	Drive Trigger control signal to protocol layer	Send in one hot space encoding	Stopstate=0 && rx_lpdt_esc = 0 && rx_ulps_esc = 0 && rx_trigger_esc = Dtat in DP and DN
RX-15	assert slave to HS mode corrupte the leder seq {00011001} in only one bit	Directed during the simulation	ErrSotHS = 1 for one cycle of RxByteClkHS && still in HS mode and get the payload data
RX-16	Randomize inputs data for 100 iterations	Randomize	accepted data

RX-17	assert slave to HS mode send right leder seq {00011000} & make negative clock skew more than one bit error	Directed during the simulation	ErrSotSyncHS = 1 for one cycle of RxByteClkHS && go to idell state
RX-18	Randomize data for 10000 iteration	Randomize	not acceptable any data
RX-19	assert Escape mode with Error in Entry CMD	Directed during the simulation	ErrEsc =1 >> untill changing in the lane state
RX-20	assert LP transmission with not a multiple eight bits	Directed during the simulation	ErrSyncEsc =1 >> is asserted untill changing in the lane state
RX-21	assert escape mode request is immediately followed by a Stop state not bridg state	Directed during the simulation	ErrControl =1 >> asserted high untill changing in the line state

3.6.2Verification Plan TX

VP_Slave Table from TX_MIPI D-PHYScope	Design Requirement Description	Stimulus Generation	Functionality Check
TX-1	Initializig the system	All signals=0	CLK_ULPSActiveNOT=1, DATA_TxUlpsActiveNot=1, DATA_DP1=z, DATA_DN1=z, clk_DP=z, clk_DN=z, the rest=0
TX-2		CLK_Enable=1, DATA_Enable=1, the rest=0	CLK_ULPSActiveNOT=1, DATA_TxUlpsActiveNot=1, DATA_DP1=1, DATA_DN1=1, clk_DP=1, clk_DN=1, CLK_StopState=1, DATA_TX_STOPSTATE=1, the rest=0
TX-3		CLK_Enable=1, DATA_Enable=1, RST=1, the rest=0	TxByteClkHS is generated, CLK_ULPSActiveNOT=1, DATA_TxUlpsActiveNot=1, DATA_DP1=1, DATA_DN1=1, clk_DP=1, clk_DN=1, CLK_StopState=1, DATA_TX_STOPSTATE=1, the rest=0
TX-4		RST=1, the rest=0	TxByteClkHS is generated , CLK_ULPSActiveNOT=1, DATA_TxUlpsActiveNot=1, DATA_DP1=z, DATA_DN1=z, clk_DP=z, clk_DN=z, the rest=0
TX-5	Testing if the reset able to handle the randomization	RST=1, randomize the rest	

TX-6		RST=0, randomize the rest	
TX-7	HS-mode entry sequence and entry command	CLK_Enable=1, DATA_Enable=1, RST=1, TxRequestHS=1, the rest=0	TxByteClkHS is generated, CLK_ULPSActiveNOT=1, DATA_TxUlpsActiveNot=1, , clk_DP ={1,0,0=>0,0,0,0,….0,=>0,0,0,1,1,1,0,1=> TxDDRClkHS_Q}, clk_DN ={1,1,0,=>1,1,1,1,,….1,=>1,1,1,0,0,0,1,0=>!TxDDRClkHS_Q}, DATA_DP1=1, DATA_DN1 =1, CLK_StopState={1,0,0=>0}, DATA_TX_STOPSTATE=1, the rest=0 => TxReadyHS=1
TX-8	Transmitting a specific data in HS_mode	CLK_Enable=1, DATA_Enable=1, RST=1, TxRequestHS=1, TxDataHS=6, the rest=0	TxByteClkHS is generated, CLK_ULPSActiveNOT=1, DATA_TxUlpsActiveNot=1, , clk_DP =TxDDRClkHS_Q, clk_DN =!TxDDRClkHS_Q, DATA_DP1={0,0,0,0,0,1,1,0}, DATA_DN1={1,1,1,1,1,0,0,1}, CLK_StopState={0}, DATA_TX_STOPSTATE={0}, the rest=0 => TxReadyHS=1
TX-9	Transmitting randomized data	CLK_Enable=1, DATA_Enable=1, RST=1, TxRequestHS=1, TxDataHS is randomized, the rest=0	

TX-10	Testing if the HS-mode will be able to handle the randomization	CLK_Enable=1, DATA_Enable=1, RST=1, TxRequestHS=1, other signals are randomized	
TX-11	Entry sequence of Esc_mode	CLK_Enable=1, DATA_Enable=1, RST=1, TxRequestEsc=1, the rest=0	TxByteClkHS is generated, CLK_ULPSActiveNOT=1, DATA_TxUlpsActiveNot=1, , clk_DP =1, clk_DN=1, DATA_DP1={1,1,0,0,0}, DATA_DN1={1,0,0,1,0}, CLK_StopState=1, DATA_TX_STOPSTATE={1,0,0,0,0}, the rest=0
TX-12	Entry command of Esc_mode	CLK_Enable=1, DATA_Enable=1, RST=1, TxRequestEsc=1, TxLpdtEsc=1, the rest=0	TxByteClkHS is generated, CLK_ULPSActiveNOT=1, DATA_TxUlpsActiveNot=1, , clk_DP =1, clk_DN=1, DATA_DP1={1,1,1,0,0,0,0,1}, DATA_DN1={0,0,0,1,1,1,1,0}, CLK_StopState=1, DATA_TX_STOPSTATE=0, the rest=0 => TxReadyEsc=1
TX-13	Transmitting a specific data	CLK_Enable=1, DATA_Enable=1, RST=1, TxRequestEsc=1, TxLpdtEsc=1, TxDataEsc=4, TxValidEsc=1, the rest=0	TxByteClkHS is generated, CLK_ULPSActiveNOT=1, DATA_TxUlpsActiveNot=1, , clk_DP =1, clk_DN=1, DATA_DP1={0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0}, DATA_DN1={1,1,1,1,1,1,1,1,1,1,0,1,1,1,1,1}, CLK_StopState=1, DATA_TX_STOPSTATE=0, the rest=0, TxReadyEsc=1

TX-14	Transmitting randomized data	CLK_Enable=1, DATA_Enable=1, RST=1, TxRequestEsc=1, TxLpdtEsc=1, TxDataEs is randomized, TxValidEsc=1, the rest=0	TxByteClkHS is generated, CLK_ULPSActiveNOT=1, DATA_TxUlpsActiveNot=1, , clk_DP =1, clk_DN=1, DATA_DP1={randomized data}, DATA_DN1={ randomized data }, CLK_StopState=1, DATA_TX_STOPSTATE=0, the rest=0, TxReadyEsc=1
TX-15	Testing if it will stay in the same mode after setting TxLpdtEsc to zero	CLK_Enable=1, DATA_Enable=1, RST=1, TxRequestEsc=1, TxDataEs is randomized, TxValidEsc=1, the rest=0	TxByteClkHS is generated, CLK_ULPSActiveNOT=1, DATA_TxUlpsActiveNot=1, , clk_DP =1, clk_DN=1, DATA_DP1={randomized data}, DATA_DN1={ randomized data }, CLK_StopState=1, DATA_TX_STOPSTATE=0, the rest=0, TxReadyEsc=1
TX-16	Testing if the Esc-mode will leave the mode if the escape request is zero	CLK_Enable=1, DATA_Enable=1, RST=1, TxLpdtEsc=1, TxValidEsc=1, the rest is randomized	TxByteClkHS is generated, CLK_ULPSActiveNOT=1, DATA_TxUlpsActiveNot=1, , clk_DP =1, clk_DN=1, DATA_DP1=1, DATA_DN1=1, CLK_StopState=1, DATA_TX_STOPSTATE=1, the rest=0
TX-17	Testing if the Esc-mode able to handle the randomization	CLK_Enable=1 DATA_Enable=1, RST=1, TxRequestEsc=1, TxLpdtEsc=1, TxValidEsc=1,	

TX-18	Trigger entry command case1	CLK_Enable=1, DATA_Enable=1, RST=1, TxRequestEsc=1, TxTriggerEsc=0001, the rest=0	TxByteClkHS is generated, CLK_ULPSActiveNOT=1, DATA_TxUlpsActiveNot=1, , clk_DP =1, clk_DN=1, DATA_DP1={0,0,1,0,1,0,0,0,0,0,0,0,1,0,0,0}, DATA_DN1={1,1,0,1,0,1,1,1,1,1,1,1,0,1,1,1}, CLK_StopState=1, DATA_TX_STOPSTATE=0, the rest=0
TX-19	Trigger entry command case2	CLK_Enable=1, DATA_Enable=1, RST=1, TxRequestEsc=1, TxTriggerEsc=0010, the rest=0	TxByteClkHS is generated, CLK_ULPSActiveNOT=1, DATA_TxUlpsActiveNot=1, , clk_DP =1, clk_DN=1, DATA_DP1={0,0,1,0,1,0,0,0,0,0,0,0,1,0,0,0}, DATA_DN1={1,1,0,1,0,1,1,1,1,1,1,1,0,1,1,1}, CLK_StopState=1, DATA_TX_STOPSTATE=0, the rest=0
TX-20	Trigger entry command case3	CLK_Enable=1, DATA_Enable=1, RST=1, TxRequestEsc=1, TxTriggerEsc=0100, the rest=0	TxByteClkHS is generated, CLK_ULPSActiveNOT=1, DATA_TxUlpsActiveNot=1, , clk_DP =1, clk_DN=1, DATA_DP1={0,0,1,0,1,0,0,0,0,0,0,0,1,0,0,0}, DATA_DN1={1,1,0,1,0,1,1,1,1,1,1,1,0,1,1,1}, CLK_StopState=1, DATA_TX_STOPSTATE=0, the rest=0
TX-21	Trigger entry command case4	CLK_Enable=1, DATA_Enable=1, RST=1, TxRequestEsc=1, TxTriggerEsc=1000, the rest=0	TxByteClkHS is generated, CLK_ULPSActiveNOT=1, DATA_TxUlpsActiveNot=1, , clk_DP =1, clk_DN=1, DATA_DP1={0,0,1,0,1,0,0,0,0,0,0,0,1,0,0,0}, DATA_DN1={1,1,0,1,0,1,1,1,1,1,1,1,0,1,1,1}, CLK_StopState=1, DATA_TX_STOPSTATE=0, the rest=0

TX_22	Testing if it will go to stopstate after changing TxTriggerEsc	CLK_Enable=1, DATA_Enable=1, RST=1, TxRequestEsc=1, TxTriggerEsc is randomized, the rest=0	TxByteClkHS is generated, CLK_ULPSActiveNOT=1, DATA_TxUlpsActiveNot=1, , clk_DP =1, clk_DN=1, DATA_DP1=1, DATA_DN1=1, CLK_StopState=1, DATA_TX_STOPSTATE=1, the rest=0
TX_23	Testing if the Trigger-mode will be able to handle the randomization	CLK_Enable=1, DATA_Enable=1, RST=1, TxRequestEsc=1, TxTriggerEsc=0001, the rest is randomized	
TX_24	Entry sequence to Ulps-mode	CLK_Enable=1, DATA_Enable=1, RST=1, TxRequestEsc=1, TxUlpsEsc=1, TxUlpsClk=1, the rest=0	CLK_ULPSActiveNOT=1=>0, DATA_TxUlpsActiveNot=1=>0, , clk_DP =0, clk_DN=0, DATA_DP1={0,0,0,1,1,1,1,0=>0}, DATA_DN1={1,1,1,0,0,0,0,1=>0} , CLK_StopState=0, DATA_TX_STOPSTATE=0, the rest=0
TX_25	Testing if it will stay in the same mode after setting TxUlpsEsc to zero	CLK_Enable=1, DATA_Enable=1, RST=1, TxRequestEsc=1, TxUlpsEsc=0, TxUlpsClk=1, the rest=0	CLK_ULPSActiveNOT=0, DATA_TxUlpsActiveNot=0, , clk_DP =0, clk_DN=0, DATA_DP1=0, DATA_DN1=0, CLK_StopState=0, DATA_TX_STOPSTATE=0, the rest=0

TX_26	Testing if the Trigger-mode will be able to handle the randomization	CLK_Enable=1, DATA_Enable=1, RST=1, TxRequestEsc=1, TxUlpsEsc=1, TxUlpsClk=1, CLK_TxULPSExit=0, DATA_TxUlpsExit=0, the rest is randomized	
TX_27	Leaving the Ulps-mode	CLK_Enable=1, DATA_Enable=1, RST=1, CLK_TxULPSExit=1, DATA_TxUlpsExit=1, the rest=0	TxByteClkHS is generated, CLK_ULPSActiveNOT=0, DATA_TxUlpsActiveNot=0, , clk_DP =0, clk_DN=0, DATA_DP1=0, DATA_DN1=0, CLK_StopState=0, DATA_TX_STOPSTATE=0, the rest=0

3.6.3Verification Plan FULL System

Scope	**Design Requirement Description**	**Stimulus Generation**	**Functionality Check**
Full_1	Assert, Enable=0	Directed during the simulation	A checker to observe the outputs to be off (all are zeros)
Full_2	Assert, Enable=1, ForceTxStopmode_M=1, ForceRxmode_M=0, ForceTxStopmode_S=0, ForceRxmode_S=1,	Directed during the simulation	A checker to observe the outputs to be as follow, Direction_M=0, Direction_S=1, Stopstate=1

Full_3	Assert, TxRequestHS=1, TxDDRClkHS-I, TxDDRClkHS-Q	Directed during the simulation	A checker to observe the outputs to be as follow, TxReady=1, Stopstate=0, TxByteClkHS, RxDDRClkHS, RxActiveHS=1, RxSyncHS=1, RxClkActiveHS=1
Full_4	Assert, TxDataHS	Rondomized data during simulation	A checker to observe the outputs to be as follow, then RxValidHS=1, RxDataHS, RxSyncHS=0
Full_5	Deassert HSmode, TxRequestHS=0	Directed during the simulation	A checker to observe the outputs to be as follow, Stopstate=1,
Full_6	Assert, TxRequstEsc_M=1, TxLpdtEsc_M=1	Directed during the simulation	A checker to observe the outputs to be as follow, Stopstate=0, RxLpdtEsc_S=1, TxReadyEsc_M=1
Full_7	Assert, TxValidEsc_M=1, TxDataEsc_M (randomized)	Rondomized data during simulation	A checker to observe the outputs to be as follow, RxClkEsc, RxDataEsc, RxValidEsc=1

Full_8	Assert, TxRequstEsc_M=1, TxLpdtEsc_M=0	Directed during the simulation	A checker to observe the outputs to be as follow, Stopstate=1, RxLpdtEsc_S=0, TxReadyEsc_M=0, RxDataEsc=0, RxDataEsc=0, RxValidEsc=0
Full_9	Assert, TxRequstEsc=1, TxTriggerEsc=1	Directed during the simulation	A checker to observe the outputs to be as follow, Stopstate=0, RxTriggerEsc_S=1
Full_10	Assert, TxRequstEsc=1, TxTriggerEsc=0	Directed during the simulation	A checker to observe the outputs to be as follow, Stopstate=1, RxTriggerEsc_S=0
Full_11	Assert, TxRequstEsc=1, TxUlpsEsc=1, TxUlpsClk=1	Directed during the simulation	A checker to observe the outputs to be as follow, Stopstate=0, UlpsActiveNot=0, RxUlpsClkNot=0, RxUlpsEsc_S=1
Full_12	Assert, TxRequstEsc=1, TxUlpsEsc=0, TxUlpsClk=0, TxUlpsExit=1	Directed during the simulation	A checker to observe the outputs to be as follow, Stopstate=1, UlpsActiveNot=1, RxUlpsClkNot=1, RxUlpsEsc_S=0

Full_13	Assert, TurnRequest_M=1	Directed during the simulation	A checker to observe the outputs to be as follow, Direction_M=1, Direction_S=0, Stopstate=0 untill the both turnaround sequences are done then Stopstate=1
Full_14	Assert, TxRequstEsc_S=1, TxLpdtEsc_S=1	Directed during the simulation	A checker to observe the outputs to be as follow, Stopstate=0, RxLpdtEsc_S=1, TxReadyEsc_M=1
Full_15	Assert, TxValidEsc_S=1, TxDataEsc_S (randomized)	Directed during the simulation	A checker to observe the outputs to be as follow, RxClkEsc, RxDataEsc, RxValidEsc=1
Full_16	Assert, TxRequstEsc_S=1, TxLpdtEsc_S=0	Directed during the simulation	A checker to observe the outputs to be as follow, Stopstate=1, RxLpdtEsc_S=0, TxReadyEsc_M=0, RxDataEsc=0, RxDataEsc=0, RxValidEsc=0
Full_17	Assert, TurnDisable_S=1	Directed during the simulation	A checker to observe the outputs to be as follow, Direction_M=0,

3.7 VERIFICATION SIMULATION RESULTS

3.7.1 RX Assertions

Assertion Name	Clock Domain	Coverage Intent	Description
HS_CLK_DIFF_ASSERT	rx_byte_clk_hs	Functional	Checks clk_dp and clk_dn are differential.
HS_DATA_DIFF_ASSERT	rx_byte_clk_hs	Functional	Checks dp and dn are differential.
HS_ACTIVE_ASSERT	rx_byte_clk_hs	State transition	Checks rx_sync_hs leads to rx_active_hs.
HS_VALID_ASSERT	rx_byte_clk_hs	State transition	Ensures rx_sync_hs leads to rx_valid_hs.
ESC_ENTRY_ASSERT	rx_clk_esc	Escape state	Checks rx_lpdt_esc is active during rx_stop_state.
ULPS_TRANSITION_ASSERT	rx_clk_esc	ULPS state	Checks rx_ulps_esc is active during rx_stop_state.
STOP_STATE_ASSERT	rx_byte_clk_hs	State hold	Checks rx_active_hs during rx_stop_state.
SOT_ERROR_ASSERT	rx_byte_clk_hs	Error detection	Detects err_sot_hs within 10 cycles of err_sot_sync_hs.

SOT_SYNC_ERROR_ASSERT	rx_byte_clk_hs	Error cleanup	Ensures err_sot_sync_hs falls after rising.
ESC_ERROR_ASSERT	rx_clk_esc	Error hold	Confirms err_esc persists during rx_stop_state.
CONTENTION_ERROR_ASSERT	rx_byte_clk_hs	Error recovery	err_contention_lp0 leads to force_rx_mode.
TRIGGER_ERROR_ASSERT	rx_clk_esc	Trigger validity	rx_trigger_esc must match predefined values.
ULPS_START_ERROR_ASSERT	rx_clk_esc	ULPS entry	ULPS should not occur while active_hs is high.
DATA_VALID_ASSERT	rx_byte_clk_hs	Data integrity	rx_data_hs should be in 8-bit range when valid.
ULPS_LPDT_ASSERT	rx_clk_esc	State separation	Ensures rx_ulps_esc and rx_lpdt_esc are mutually exclusive.
HS_EXIT_ASSERT	rx_byte_clk_hs	State transition	Verifies transition from active_hs to stop_state after HS transmission.

LP_MODE_ENTRY_ASSERT	rx_clk_esc	Mode switch	Confirms LPDT and stop_state during LP mode entry.
SYNC_EDGE_DETECT_ASSERT	rx_byte_clk_hs	Timing	Ensures sync_hs is detected on rising clock edge only.
ULPS_EXIT_ASSERT	rx_clk_esc	State transition	Checks that exiting ULPS clears rx_ulps_esc and activates rx_lpdt_esc.
ERROR_RECOVERY_ASSERT	rx_clk_esc	Fault tolerance	Ensures system returns to idle after error detection.
RX_MODE_ASSERT	rx_clk_esc	Config check	Ensures force_rx_mode is stable during LP0 contention.
HS_DATA_VALID_DELAY_ASSERT	rx_byte_clk_hs	Timing constraint	Data must be valid within 3 cycles after active_hs.
ULPS_TIMEOUT_ASSERT	rx_clk_esc	Timeout	ULPS mode should not exceed predefined cycle timeout.

Name	Assertion Type	Language	Enable	Failure Count	Pass Count	Active Count	Memory	Peak Memory	Peak Memory Time	Cur	ATV	Assertion Expression
/D_PHY_RX_TB/ASSERT_1/a_ULPS	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(posedge D_PHY_DUT.DataLaneDu
/D_PHY_RX_TB/ASSERT_1/a_Reserved	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(posedge D_PHY_DUT.DataLaneDu
/D_PHY_RX_TB/ASSERT_1/a_LPDT2	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(posedge D_PHY_DUT.DataLaneDu
/D_PHY_RX_TB/ASSERT_1/assert__A1	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(hs_clk) D_PHY_DUT.ClkLaneDut.SI
/D_PHY_RX_TB/ASSERT_1/assert__A2	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(posedge assert_clk) (LP11)|->1 #
/D_PHY_RX_TB/ASSERT_1/assert__HS_TERMENATION_ENABLE	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(posedge assert_clk) disable iff (D_
/D_PHY_RX_TB/ASSERT_1/assert__CLK_SETTLE_ENABLE	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(assert_clk) disable iff (D_PHY_DU
/D_PHY_RX_TB/ASSERT_1/assert__rx_clk_active	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(hs_clk) disable iff (D_PHY_DUT.Cl
/D_PHY_RX_TB/ASSERT_1/assert__hs_leaving_entry	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(posedge assert_clk) disable iff (D_
/D_PHY_RX_TB/ASSERT_1/assert__ULPS_Flags	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(posedge assert_clk) disable iff (D_
/D_PHY_RX_TB/ASSERT_1/assert__A3_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(posedge assert_clk) disable iff (D_
/D_PHY_RX_TB/ASSERT_1/assert__A5_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(posedge assert_clk) disable iff (D_
/D_PHY_RX_TB/ASSERT_1/assert__A6_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(posedge assert_clk) disable iff (D_
/D_PHY_RX_TB/ASSERT_1/assert__A7_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(hs_clk) disable iff (D_PHY_DUT.Cl
/D_PHY_RX_TB/ASSERT_1/assert__A8_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(hs_clk) disable iff (D_PHY_DUT.Cl
/D_PHY_RX_TB/ASSERT_1/assert__A9_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(hs_clk) disable iff (D_PHY_DUT.Cl
/D_PHY_RX_TB/ASSERT_1/assert__A10_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(hs_clk) disable iff (D_PHY_DUT.Cl
/D_PHY_RX_TB/ASSERT_1/assert__A11_p	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(hs_clk) disable iff (D_PHY_DUT.Cl
/D_PHY_RX_TB/ASSERT_1/assert__A66	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(hs_clk) disable iff (D_PHY_DUT.Cl
/D_PHY_RX_TB/ASSERT_1/assert__A77	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(hs_clk) disable iff (D_PHY_DUT.Cl
/D_PHY_RX_TB/ASSERT_2/assert__SYNC_VALID_SEQ_1	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(posedge D_PHY_DUT.DataLaneDu
/D_PHY_RX_TB/ASSERT_2/assert__SYNC_VALID_SEQ_2	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(posedge D_PHY_DUT.DataLaneDu
/D_PHY_RX_TB/ASSERT_2/assert__SYNC_VALID_SEQ_ONE_ERROR_1	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(posedge D_PHY_DUT.DataLaneDu
/D_PHY_RX_TB/ASSERT_2/assert__SYNC_VALID_SEQ_ONE_ERROR_2	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(posedge D_PHY_DUT.DataLaneDu
/D_PHY_RX_TB/ASSERT_2/assert__SYNC_VALID_SEQ_ONE_ERROR_3	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(posedge D_PHY_DUT.DataLaneDu
/D_PHY_RX_TB/ASSERT_2/assert__SYNC_VALID_SEQ_TWO_ERRORS	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(posedge D_PHY_DUT.DataLaneDu
/D_PHY_RX_TB/ASSERT_2/assert__HS_EXIT	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(posedge D_PHY_DUT.DataLaneDu
/D_PHY_RX_TB/ASSERT_2/assert__DATA_VALID_1	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(posedge D_PHY_DUT.DataLaneDu
/D_PHY_RX_TB/ASSERT_2/assert__DATA_VALID_2	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(posedge D_PHY_DUT.DataLaneDu
/D_PHY_RX_TB/ASSERT_2/assert__TRAILER_DONE	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(D_PHY_DUT.DataLaneDut_2.HS_E
/D_PHY_RX_TB/ASSERT_2/assert__FULL_FIFO	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(posedge D_PHY_DUT.DataLaneDu
/D_PHY_RX_TB/ASSERT_2/a_Reset_Trigger	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(posedge D_PHY_DUT.DataLaneDu
/D_PHY_RX_TB/ASSERT_2/a_Trigger_4	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(posedge D_PHY_DUT.DataLaneDu
/D_PHY_RX_TB/ASSERT_2/a_ULPS	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(posedge D_PHY_DUT.DataLaneDu
/D_PHY_RX_TB/ASSERT_2/a_Reserved	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(posedge D_PHY_DUT.DataLaneDu
/D_PHY_RX_TB/ASSERT_2/a_LPDT2	Concurrent	SVA	on	0	1	-	0B	0B	0 ps	0	off	assert(@(posedge D_PHY_DUT.DataLaneDu

Figure 3.16: RX Assertions

3.7.2 TX Assertions

Name	Language	Enal	Log	Count	AtLeast	Limit	Weight	Cmplt %	Cmplt graph	Include
/master_top/DUT/SVA/cover__TxUlpsEsc	SVA	✓	Off	62857	1	Unlimited	1	100%		✓
/master_top/DUT/SVA/cover__TxLpdtEsc	SVA	✓	Off	05487	1	Unlimited	1	100%		✓
/master_top/DUT/SVA/cover__TxLpdtEsc_on	SVA	✓	Off	3	1	Unlimited	1	100%		✓
/master_top/DUT/SVA/cover__TxRequestEsc_off_TxRequestHS_on	SVA	✓	Off	216	1	Unlimited	1	100%		✓
/master_top/DUT/SVA/cover__TxRequestEsc_on_TxRequestHS_off	SVA	✓	Off	318	1	Unlimited	1	100%		✓
/master_top/DUT/SVA/cover__TxRequestEsc_clk	SVA	✓	Off	29454	1	Unlimited	1	100%		✓
/master_top/DUT/SVA/cover__TxRequestHS_TxReadyHS	SVA	✓	Off	2	1	Unlimited	1	100%		✓
/master_top/DUT/SVA/cover__TxRequestHS_negclk_off	SVA	✓	Off	158	1	Unlimited	1	100%		✓
/master_top/DUT/SVA/cover__TxRequestHS_posclk_off	SVA	✓	Off	164	1	Unlimited	1	100%		✓
/master_top/DUT/SVA/cover__TxRequestHS_negclk_on	SVA	✓	Off	54	1	Unlimited	1	100%		✓
/master_top/DUT/SVA/cover__TxRequestHS_posclk_on	SVA	✓	Off	66	1	Unlimited	1	100%		✓

Figure 3.17: TX Assertions

3.7.3 Functional Coverage

Functional Coverage RX Table

Coverage Group	Description	Main Coverpoints / Cross Coverage
cg_mode_coverage	Covers main operating modes and transitions between LPDT, HS, ULPS, and Stop State conditions.	rx_active_hs, rx_lpdt_esc, rx_ulps_esc, stop_state, cross(lp_mode, ulps_mode), cross(ulps_mode, stop_state)
cg_error_coverage	Tracks different types of protocol-level error conditions and their relation to operating modes.	err_sot_hs, err_sot_sync_hs, err_esc, err_sync_esc, err_control, err_contention_lp0, err_contention_lp1, cross(sot_error, hs_mode), cross(esc_error, ulps_mode), cross(contention_lp0, contention_lp1), cross(sot_error, sync_error, esc_error)
cg_lane_coverage	Captures RX lane activity including data values and valid strobe. Ensures byte values are covered during HS mode.	rx_valid_hs, rx_data_hs [bins for values], cross(rx_valid_hs, rx_data_hs)
cg_ulps_transitions	Covers ULPS entry and exit transitions to ensure correct behavior of the protocol power-saving state.	ULPS entry (rx_ulps_esc rising edge), ULPS exit (rx_ulps_esc falling edge)

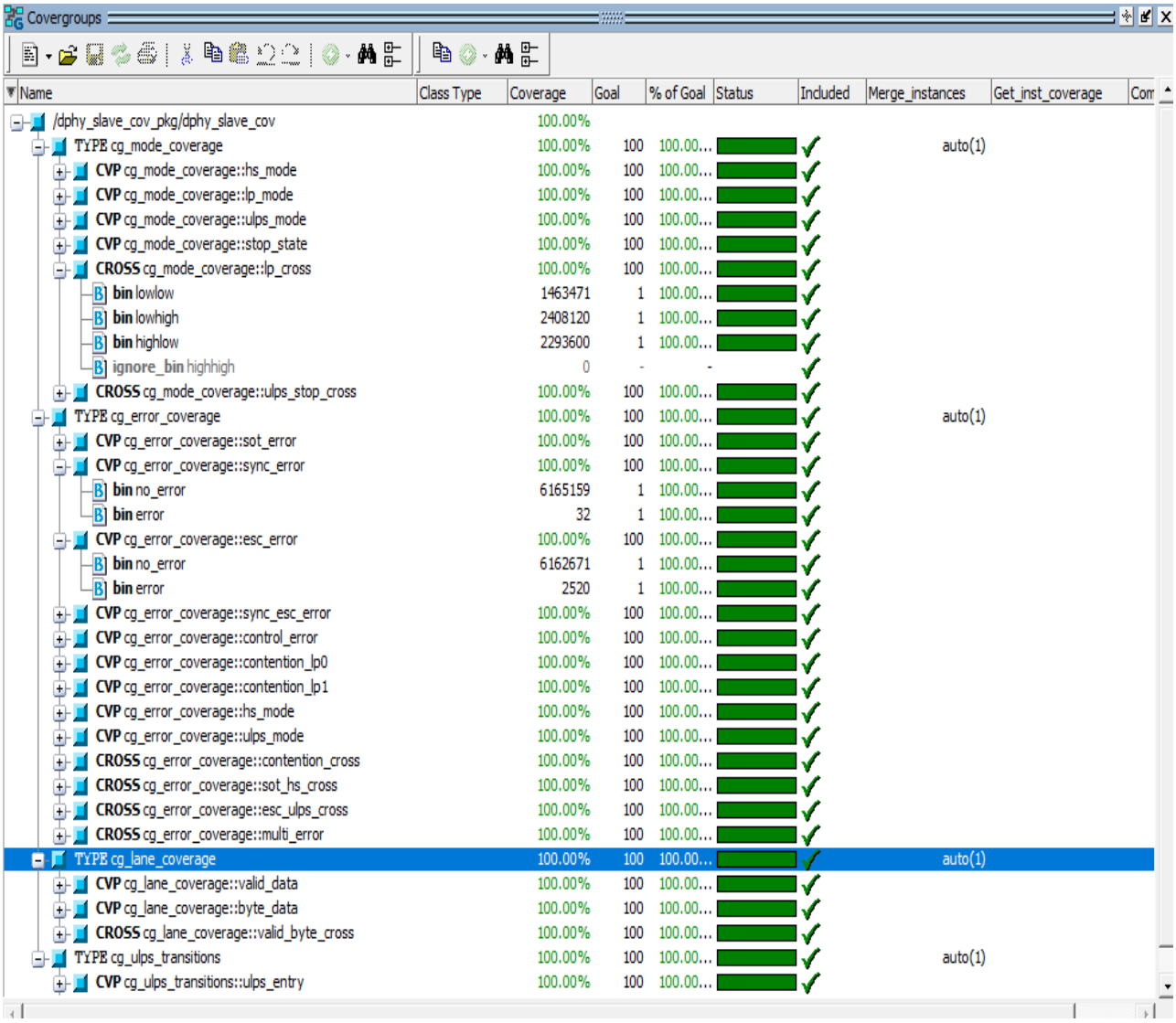

Covergroups
Name
Class Type
Coverage
Goal
% of Goal
Status
Included
Merge_instances
Get_inst_coverage
/dphy_slave_cov_pkg/dphy_slave_cov 100.00%
TYPE cg_mode_coverage 100.00% 100 100.00... auto(1)
CVP cg_mode_coverage::hs_mode 100.00% 100 100.00...
CVP cg_mode_coverage::lp_mode 100.00% 100 100.00...
CVP cg_mode_coverage::ulps_mode 100.00% 100 100.00...
CVP cg_mode_coverage::stop_state 100.00% 100 100.00...
CROSS cg_mode_coverage::lp_cross 100.00% 100 100.00...
bin lowlow 1463471 1 100.00...
bin lowhigh 2408120 1 100.00...
bin highlow 2293600 1 100.00...
ignore_bin highhigh 0 - -
CROSS cg_mode_coverage::ulps_stop_cross 100.00% 100 100.00...
TYPE cg_error_coverage 100.00% 100 100.00... auto(1)
CVP cg_error_coverage::sot_error 100.00% 100 100.00...
CVP cg_error_coverage::sync_error 100.00% 100 100.00...
bin no_error 6165159 1 100.00...
bin error 32 1 100.00...
CVP cg_error_coverage::esc_error 100.00% 100 100.00...
bin no_error 6162671 1 100.00...
bin error 2520 1 100.00...
CVP cg_error_coverage::sync_esc_error 100.00% 100 100.00...
CVP cg_error_coverage::control_error 100.00% 100 100.00...
CVP cg_error_coverage::contention_lp0 100.00% 100 100.00...
CVP cg_error_coverage::contention_lp1 100.00% 100 100.00...
CVP cg_error_coverage::hs_mode 100.00% 100 100.00...
CVP cg_error_coverage::ulps_mode 100.00% 100 100.00...
CROSS cg_error_coverage::contention_cross 100.00% 100 100.00...
CROSS cg_error_coverage::sot_hs_cross 100.00% 100 100.00...
CROSS cg_error_coverage::esc_ulps_cross 100.00% 100 100.00...
CROSS cg_error_coverage::multi_error 100.00% 100 100.00...
TYPE cg_lane_coverage 100.00% 100 100.00... auto(1)
CVP cg_lane_coverage::valid_data 100.00% 100 100.00...
CVP cg_lane_coverage::byte_data 100.00% 100 100.00...
CROSS cg_lane_coverage::valid_byte_cross 100.00% 100 100.00...
TYPE cg_ulps_transitions 100.00% 100 100.00... auto(1)
CVP cg_ulps_transitions::ulps_entry 100.00% 100 100.00...

Figure 3.18: RX Functional Coverage

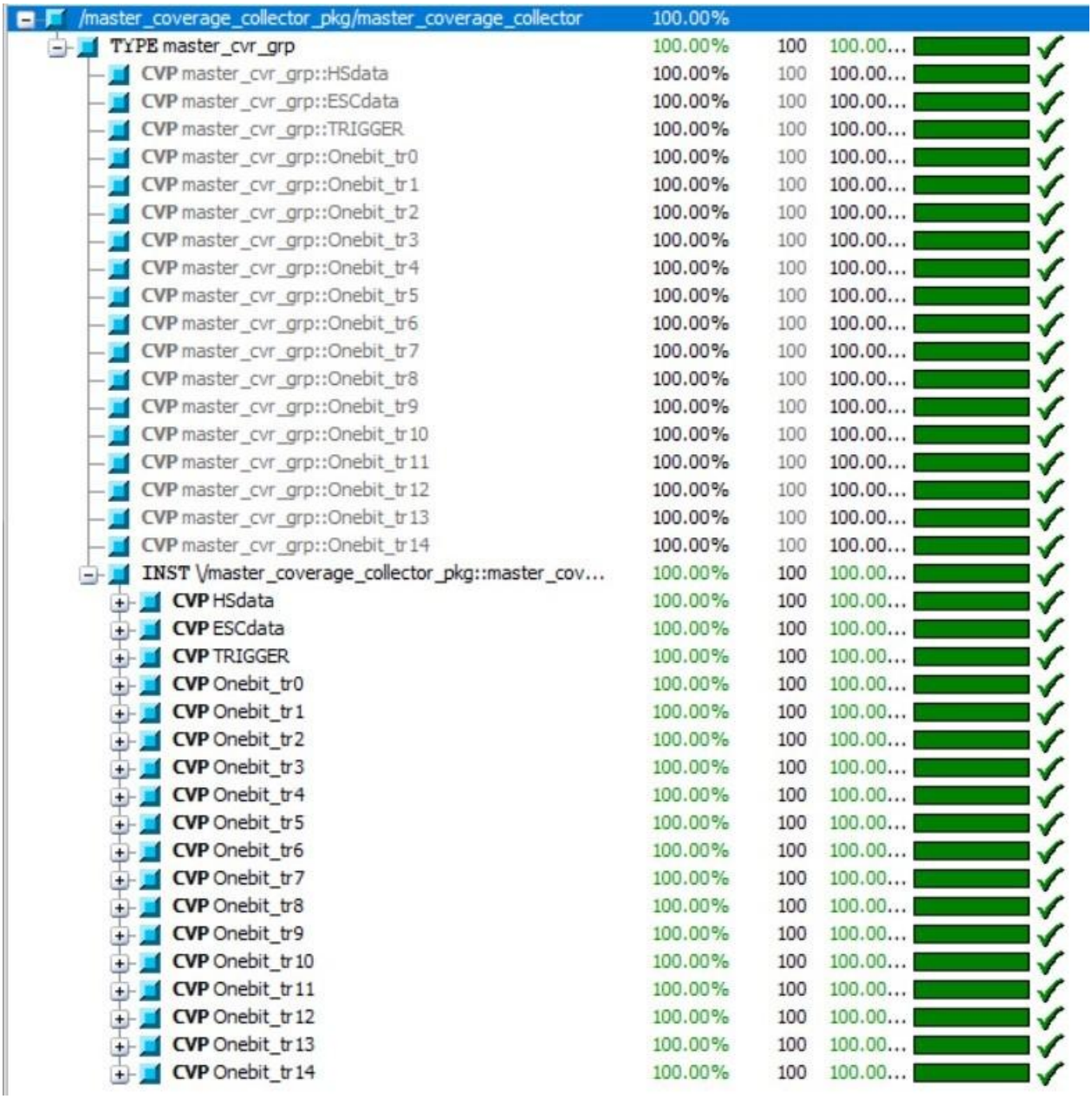


Figure 3.19: TX Functional Coverage

3.7.4 Code Coverage:

```
TOTAL COVERGROUP COVERAGE: 100.00%  COVERGROUP TYPES: 4

TOTAL DIRECTIVE COVERAGE: 95.00%  COVERS: 20

TOTAL ASSERTION COVERAGE: 95%  ASSERTIONS: 30

Total Coverage By File code coverage only: 94.05%
```

Figure 3.20: RXPHY Code Coverage

TOTAL COVERGROUP COVERAGE: 85.71% COVERGROUP TYPES: 1

TOTAL DIRECTIVE COVERAGE: 100.00% COVERS: 11

Total Coverage By Instance (filtered view): 75.94%

Directives	11	11	0	1	100.00%
Expressions	35	35	0	1	100.00%
FSM States	47	44	3	1	93.61%
Toggles	1204	874	330	1	72.59%

Figure 3.21: TXPHY Code Coverage

CHAPTER 4: RESULTS

4.1 BEHAVIORAL SIMULATION RESULT

4.1.1 High Speed Mode Proper Operation

Before transmission begins, The Clock Lane in MIPI D-PHY plays a critical role in synchronizing High-Speed (HS) data transmission. It is **unidirectional** and always driven by the **Master** side, delivering a **DDR (Double Data Rate) clock** that acts as a timing reference for the **Slave** receiver. Unlike data lanes, the Clock Lane never changes direction and does not participate in Escape mode or Reverse communication. To initiate clocking for data reception:

- The master drives **LP-01** on the Clock Lane to indicate the start of High-Speed transmission.
- After TLPX, the master transitions to **LP-00**, initiating the **Bridge state**.
- Then, the High-Speed drivers are enabled, and the lane enters HS mode by driving a differential signal (HS-0).
- Finally, the master starts toggling the differential signal between HS-0 and HS-1 continuously, which forms the **DDR Clock**.

After sending high speed clock lane entry sequence the clock lane send an indication using data_lane_start signal to data lane to start sending the data using DDR clock in high speed mode.

To initiate HS transmission, the master transitions the data lane through a sequence of Low-Power states:

- The master drives **LP-01 (HS Request)** for a duration of TLPX.
- It then switches to **LP-00 (Bridge)** for a duration of THS-PREPARE.
- After preparing the line, the master enables the High-Speed transmitter and disables the Low-Power drivers.
- The line is then driven with a constant **HS-0** differential level for THS-ZERO.
- A synchronization pattern (typically 00011101) is sent to allow the receiver to align.

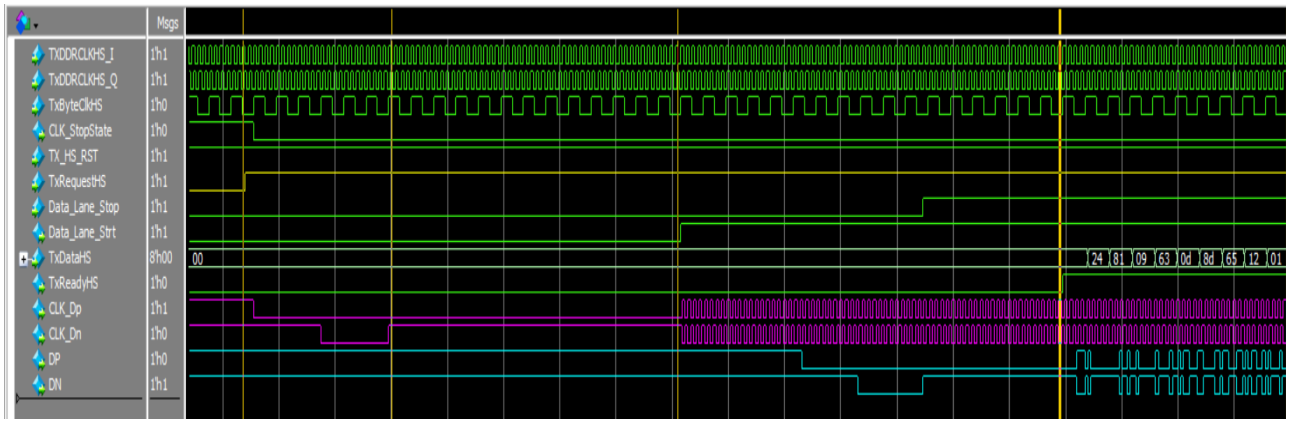


Figure 4.1: Master high speed entry sequence

Once the SoT sequence begins:

- The slave detects the LP-11 → LP-01 → LP-00 transition on the clock lane.
- After a specified duration (TD-TERM-EN), the slave enables its termination to prepare for High-Speed mode.
- It then waits for THS-SETTLE to ensure stable clock reception.
- Once toggling is detected, the slave locks onto the incoming DDR clock signal

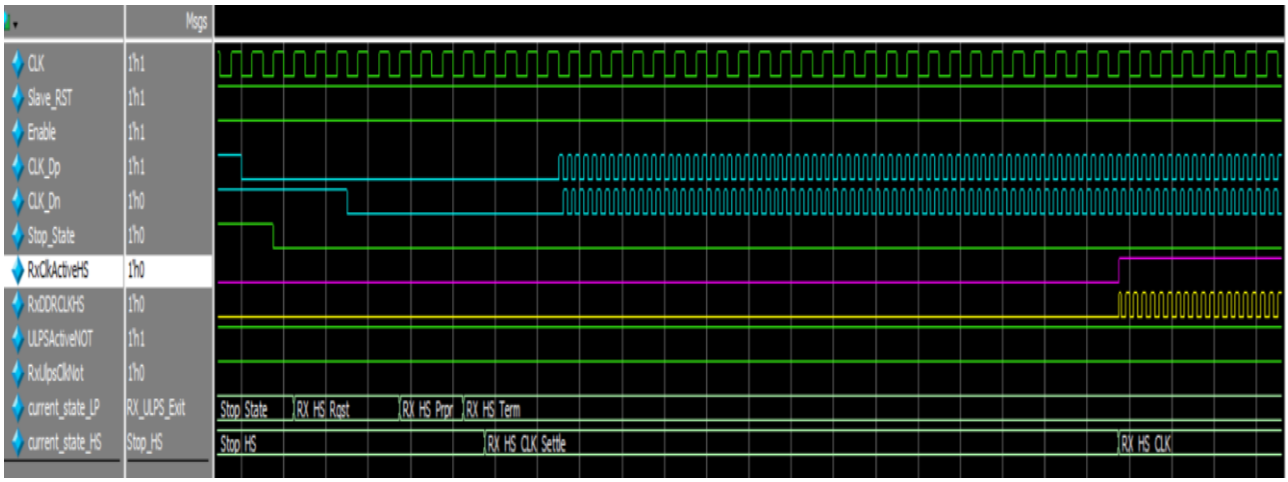


Figure 4.2: Slave clock lane high speed entry sequence

Once synchronization is established:

- The master begins transmitting data in byte format via TxDataHS, synchronized with the rising edge of TxByteClkHS.
- The slave, now in HS receive mode, asserts RxActiveHS to indicate ongoing reception and monitors RxSyncHS to detect the beginning of valid data.
- Data is sampled and presented on RxDataHS, with RxValidHS indicating byte-level validity.

Data transmission continues uninterrupted in burst mode, with no allowance for pausing. The number of bytes transmitted must be an integer value and at least one byte in length.

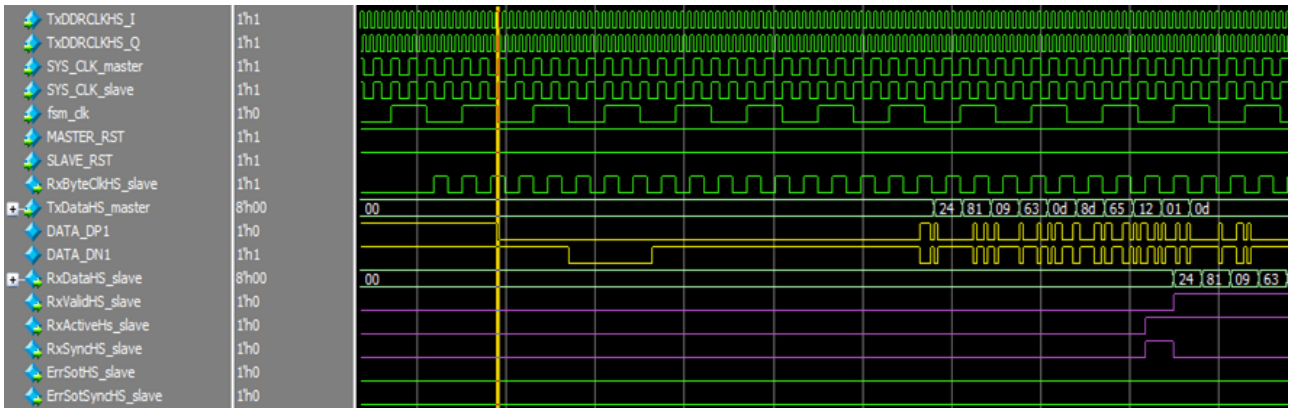


Figure 4.3: Slave data lane high speed entry sequence

The master ends the burst as follows:

- It continues driving the data lane with valid HS signals for a duration of THS-TRAIL, ensuring proper termination.
- Then, it disables the HS transmitter and re-enables the LP drivers.
- The line is returned to **LP-11 (Stop state)**, marking the end of transmission.

After the data burst ends on the data lanes:

- The clock continues toggling briefly to allow the slave to latch the final bits.
- The master then disables the HS drivers on the Clock Lane and re-enters **LP-11**.
- The clock line returns to Low-Power Stop state and remains idle until the next transmission is initiated.

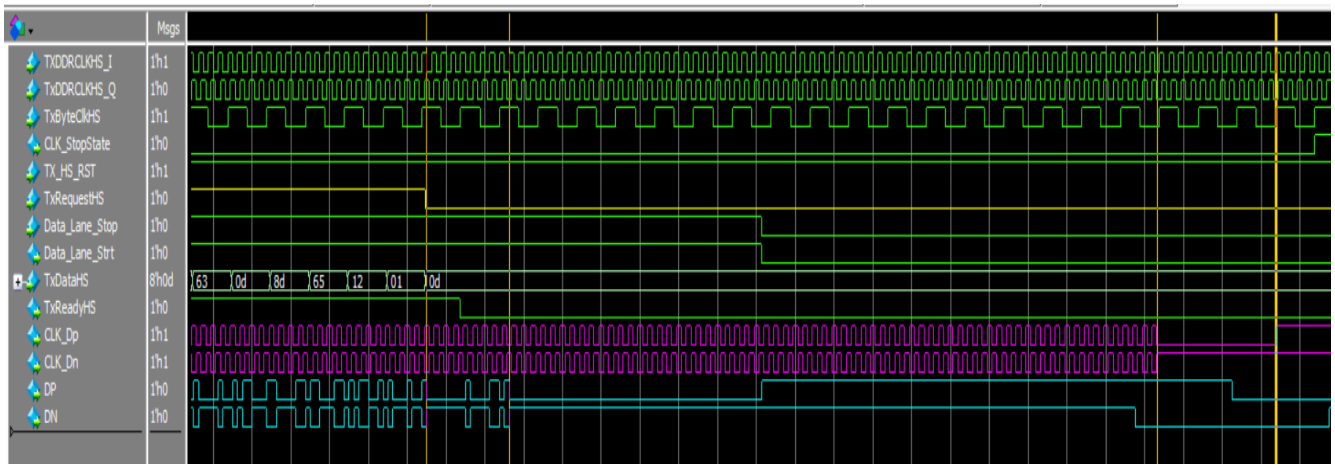


Figure 4.4: Master high speed exit sequence

The slave data lane detects the return to LP-11 and accordingly deasserts RxValidHS and RxActiveHS, indicating the completion of data reception.

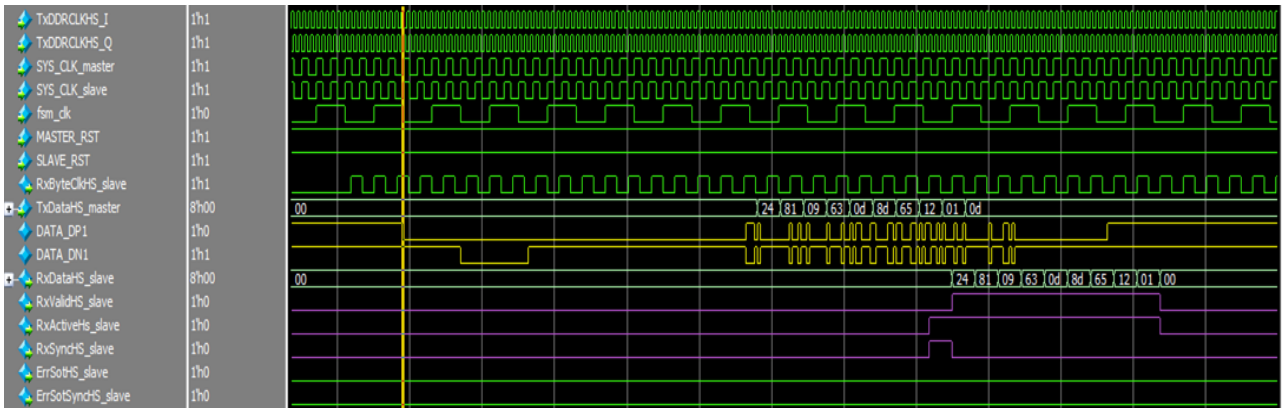


Figure 4.5: Slave data lane high speed Exit sequence

The slave clock lane detects the return to LP-11 and deactivates its clock receiver, completing the cycle.

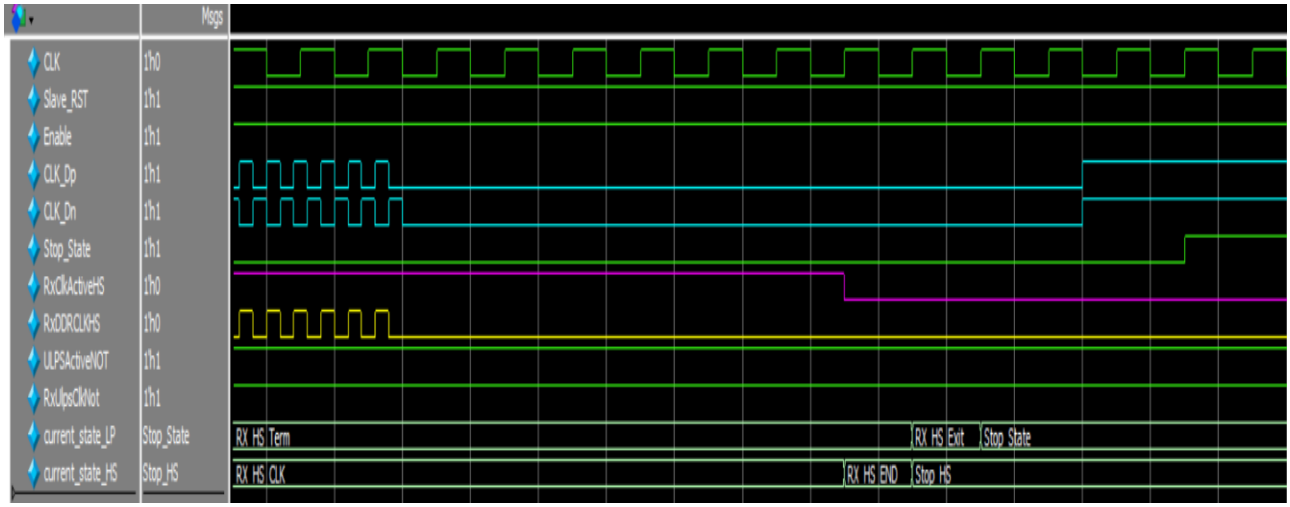


Figure 4.6: Slave clock lane high speed Exit sequence

4.1.2 Escape Mode Proper Operation

1. Low power data transmission Mode (Escape Mode)

To begin Low-Power data transmission:

- The master transitions from LP-11 to LP-10 (Mark-1) to LP-00 to LP-01, then to LP-00 (Bridge state), to signal the start of transmission.
- The protocol layer asserts TxValidLP = 1 to indicate it is sending valid LP data to the PHY.
- The PHY responds with TxReadyLP = 1 when it's ready to begin transmission on the LP lines.

As data is transmitted:

- The slave's LP-RX circuits continuously monitor the Dp and Dn lines.
- They detect and decode the single-ended logic levels driven by the master.
- The data is sampled based on signal level changes, not clock edges (as LPDT is asynchronous).
- Each stable state on the line for TLPX is interpreted as a valid bit.

After all bits are sent:

- The master transitions the line back to LP-11 (Stop state) by driving both Dp and Dn high.
- This return to Stop state signals the slave that the LPDT transmission is complete.
- The LP-RX at the slave interprets this as the end of the data block and resumes monitoring for the next sequence.

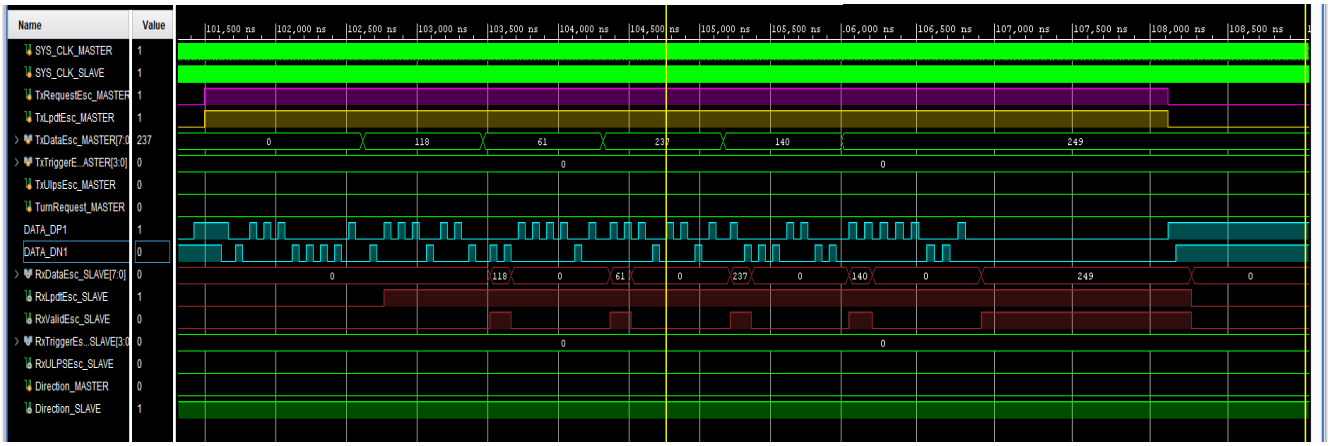


Figure 4.7: Low power data transmission

2. Trigger Mode (Escape Mode)

To begin a Trigger transmission:
• The master transitions from LP-11 to LP-10, then to LP-00, LP-01, and back to LP-00 to enter Escape Mode.
• This escape entry pattern is recognized by the slave as the start of a control command.

Once in Escape Mode:
• The master sends an 8-bit trigger command using spaced-one-hot encoded bits.
• Examples of trigger codes include:
  – Trigger 0: 01100010
  – Trigger 1: 01011101
  – Trigger 2: 00100001
  – Trigger 3: 10100000

While the command is being sent:
• The slave monitors the LP lines and decodes the incoming waveform.
• Upon recognizing a valid trigger command, the corresponding RxTriggerEsc[n] signal is asserted.

After the command is sent:
• The master returns both Dp and Dn to high, transitioning back to LP-11.
• This signals the end of the escape session and deactivates the trigger on the slave side.
• The receiver resets its trigger detection and resumes idle monitoring.

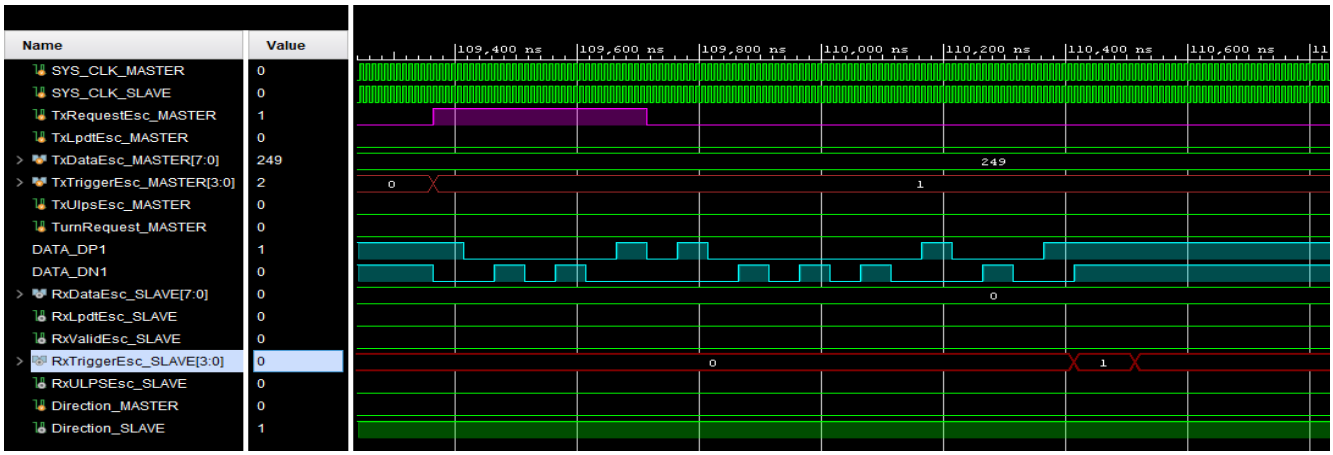


Figure 4.8: trigger operation

3. Ultra Low Power State Mode (Escape Mode)

To enter ULPS:

• the master transitions the data or clock lane from LP-11 to LP-10, then to LP-00, LP-01, and finally returns to LP-00 to enter Escape Mode.

• After entering Escape Mode, the master drives both Dp and Dn lines low (LP-00) continuously.

• This prolonged LP-00 state signals the receiver to enter the Ultra Low Power State.

While in ULPS:

• Both the master and slave turn off their internal circuit blocks for that lane to save power.

• The receiver stays in ULPS until it detects a proper exit sequence.

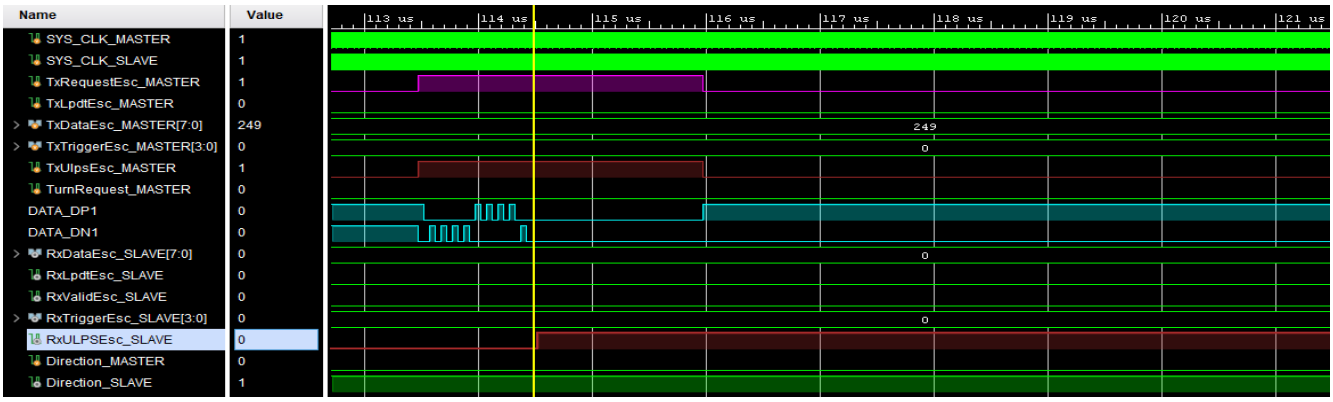


Figure 4.9: ULPS Entry

To exit ULPS:

• The master drives the LP-01 state (Dn high, Dp low) for at least TULPSEXIT duration.

• This triggers the receiver to begin ULPS exit and reactivate its circuitry.

• After the exit pattern, the lane returns to LP-11 (Stop state), completing the ULPS exit.

Figure 2.10: ULPS Exit

4.1.3 Turnaround (LP Mode Bidirectional Control)

To initiate turnaround:
• The master drives the data lane in LP-11, then releases control by stopping all driving activity.
• The line remains in LP-11 (both Dp and Dn high), but the master's LP drivers are disabled.
• This idle condition signals to the slave that the bus is now free for reverse transmission.

On the slave side:
• The slave detects that the master has released control of the bus.
• After a defined delay (TT-TA-SURE), the slave enables its LP-TX drivers.
• The slave then begins its own transmission by driving the LP lines (e.g., for response or interrupt signaling).
• Transmission follows standard LPDT rules, starting with LP-10 → LP-00 and continuing with valid bit patterns.

During this reverse transmission:
• The master switches its LP-RX circuits ON and starts monitoring the LP lines.
• It receives and decodes the data bits asynchronously based on the line levels.
• Once the slave finishes its transmission, it returns the lane to LP-11.

To return control back to the master:
• The slave disables its LP drivers after reaching LP-11.
• The master detects the bus is free again and may resume driving if needed.

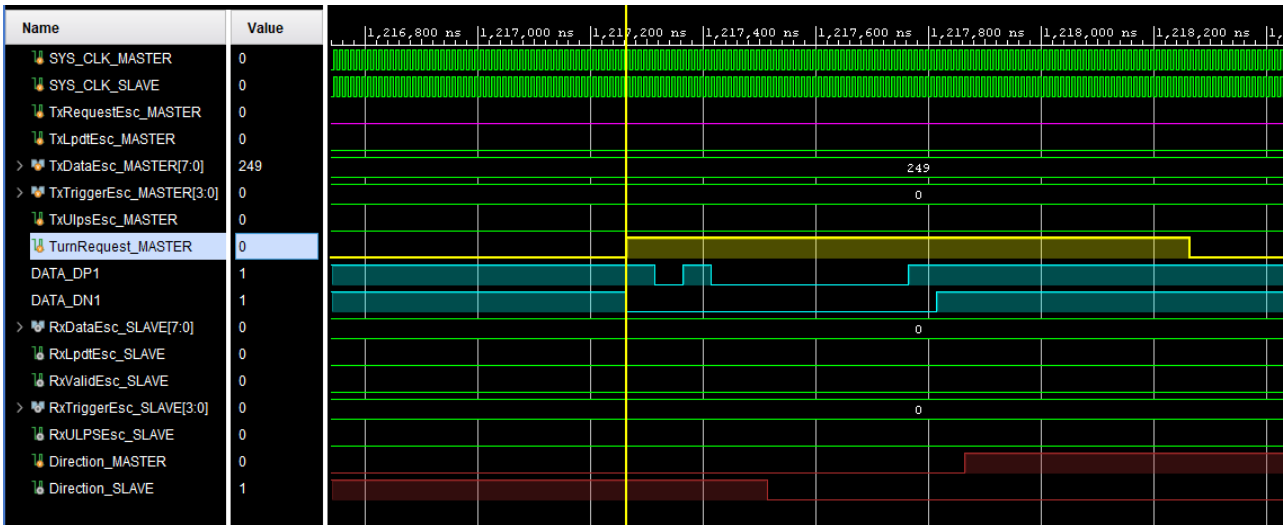


Figure 4.11: TurnAround operation

4.2 SYNTHESIS RESULTS

We used to do synthesis using Synopsys Design Compiler.

4.2.1 Master Side

4.2.1.1 Constraints

Constraints	Value
Period TxDDRClkHS-I	5ns(200Mhz)
Period TxDDRClkHS-Q	5ns(200Mhz) Phase (90)
Clock Setup Skew	0.5ns
Clock Hold Skew	0.2ns
Input Transition Time	0.1ns
Input Delay	0.5ns
Output Delay	0.5ns
Output Load Capacitance	0.15pf

Table 4.1: Constraints for Master Side DDR_CLK

Constraints	Value
Period SYS_CLK	10ns(100Mhz)
Clock Setup Skew	0.5ns
Clock Hold Skew	0.2ns
Input Transition Time	0.2ns
Input Delay	2ns
Output Delay	2ns
Output Load Capacitance	0.15pf

Table 4.2: Constraints for Master Side DDR_CLK

4.2.1.2 Block Diagram Resulted

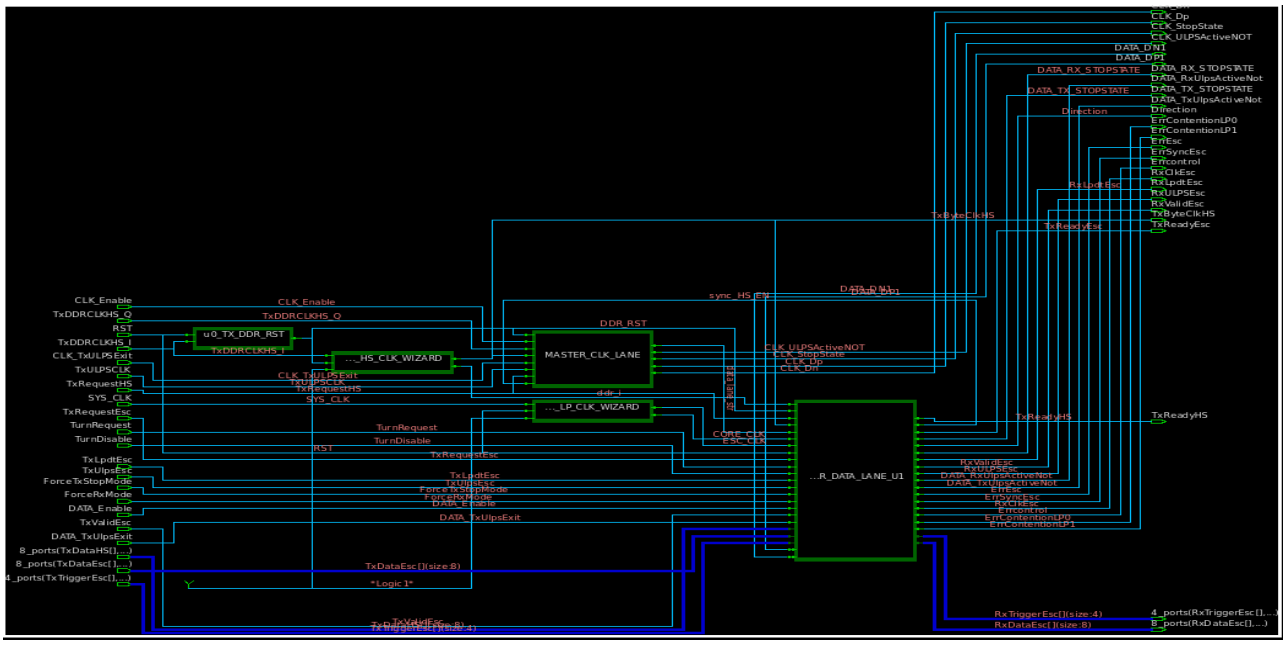


Figure 4.12: Block diagram resulted for Master Side

4.2.1.3 Timing Reports Resulted

```
****************************************
Report : constraint
         -all_violators
Design : MASTER
Version: K-2015.06
Date   : Sun May 11 03:14:18 2025
****************************************

This design has no violated constraints.
```

```
Design Rules
-----------------------------------
Total Number of Nets:          1212
Nets With Violations:             0
Max Trans Violations:             0
Max Cap Violations:               0
-----------------------------------
```

Figure 4.13: Constraints report for Master Side

```
****************************************
Report : clocks
Design : MASTER
Version: K-2015.06
Date   : Sun May 11 03:14:18 2025
****************************************

Attributes:
    d - dont_touch_network
    f - fix_hold
    p - propagated_clock
    G - generated_clock
    g - lib_generated_clock

Clock          Period   Waveform            Attrs     Sources
------------------------------------------------------------------------------
MASTER_CORE_CLK
                10.00   {0 5}               G d       {MASTER_LP_CLK_WIZARD/o_ref_clk}
MASTER_ClkEsc   50.00   {0 25}              G d       {MASTER_LP_CLK_WIZARD/o_div_clk}
SYS_CLK         10.00   {0 5}               d         {SYS_CLK}
TxByteClkHS     20.00   {0 10}              G d       {MASTER_HS_CLK_WIZARD/o_div_clk}
TxClkHS_DDR      5.00   {0 2}               G d       {MASTER_HS_CLK_WIZARD/o_ref_clk}
TxDDRCLKHS_I     5.00   {0 2}               d         {TxDDRCLKHS_I}
TxDDRCLKHS_Q     5.00   {1.25 3.75}         d         {TxDDRCLKHS_Q}
ddr_q_mux        5.00   {1.25 3.75}         G         {MASTER_CLK_LANE/U_MASTER_MUX_OUT/DP_mux}
ddr_q_txhs       5.00   {1.25 3.75}         G d       {MASTER_CLK_LANE/U_HS/HS_OUT}
------------------------------------------------------------------------------

Generated     Master         Generated      Master         Waveform
Clock         Source         Source         Clock          Modification
------------------------------------------------------------------------------
MASTER_CORE_CLK
              SYS_CLK        {MASTER_LP_CLK_WIZARD/o_ref_clk}
                                            SYS_CLK        divide_by(1)
MASTER_ClkEsc SYS_CLK        {MASTER_LP_CLK_WIZARD/o_div_clk}
                                            SYS_CLK        divide_by(5)
TxByteClkHS   TxDDRCLKHS_I   {MASTER_HS_CLK_WIZARD/o_div_clk}
                                            TxDDRCLKHS_I   divide_by(4)
TxClkHS_DDR   TxDDRCLKHS_I   {MASTER_HS_CLK_WIZARD/o_ref_clk}
                                            TxDDRCLKHS_I   divide_by(1)
ddr_q_mux     MASTER_CLK_LANE/U_HS/HS_OUT
                             {MASTER_CLK_LANE/U_MASTER_MUX_OUT/DP_mux}
                                            ddr_q_txhs     divide_by(1)
ddr_q_txhs    TxDDRCLKHS_Q   {MASTER_CLK_LANE/U_HS/HS_OUT}
                                            TxDDRCLKHS_Q   divide_by(1)
------------------------------------------------------------------------------
```

Figure 4.14: Clocks report for Master Side

4.2.1.4 Power Reports Resulted

Hierarchy	Switch Power	Int Power	Leak Power	Total Power	%
MASTER	5.41e-02	0.213	6.25e+06	0.273	100.0

Figure 4.15: Power report for Master Side

4.2.1.5 Area Reports Resulted

Area

Combinational Area:	6209.445994
Noncombinational Area:	4769.165054
Buf/Inv Area:	616.590818
Total Buffer Area:	29.42
Total Inverter Area:	587.17
Macro/Black Box Area:	0.000000
Net Area:	0.000000
Cell Area:	10978.611048
Design Area:	10978.611048

Cell Count

Hierarchical Cell Count:	53
Hierarchical Port Count:	828
Leaf Cell Count:	1092
Buf/Inv Cell Count:	172
Buf Cell Count:	6
Inv Cell Count:	166
CT Buf/Inv Cell Count:	0
Combinational Cell Count:	911
Sequential Cell Count:	181
Macro Count:	0

Figure 4.16: Area report for Master Side

4.2.2 Slave Side

4.2.2.1 Constraints

Constraints	Value
Period SYS_CLK	10ns(100Mhz)
Clock Setup Skew	1ns
Clock Hold Skew	0.05ns
Input Transition Time	0.3ns
Input Delay	0.1ns
Output Delay	0.1ns
Output Load Capacitance	0.15pf

Table 4.3: Constraints for Slave Side

Constraints	Value
Period Dp_CLK (Received from TX)	5ns(200Mhz)
Clock Setup Skew	0.5ns
Clock Hold Skew	0.05ns
Input Transition Time	0.1ns
Input Delay	0.05ns
Output Delay	0.05ns
Output Load Capacitance	0.15pf

Table 4.4: Constraints for Slave Side

4.2.2.2 Block Diagram Resulted

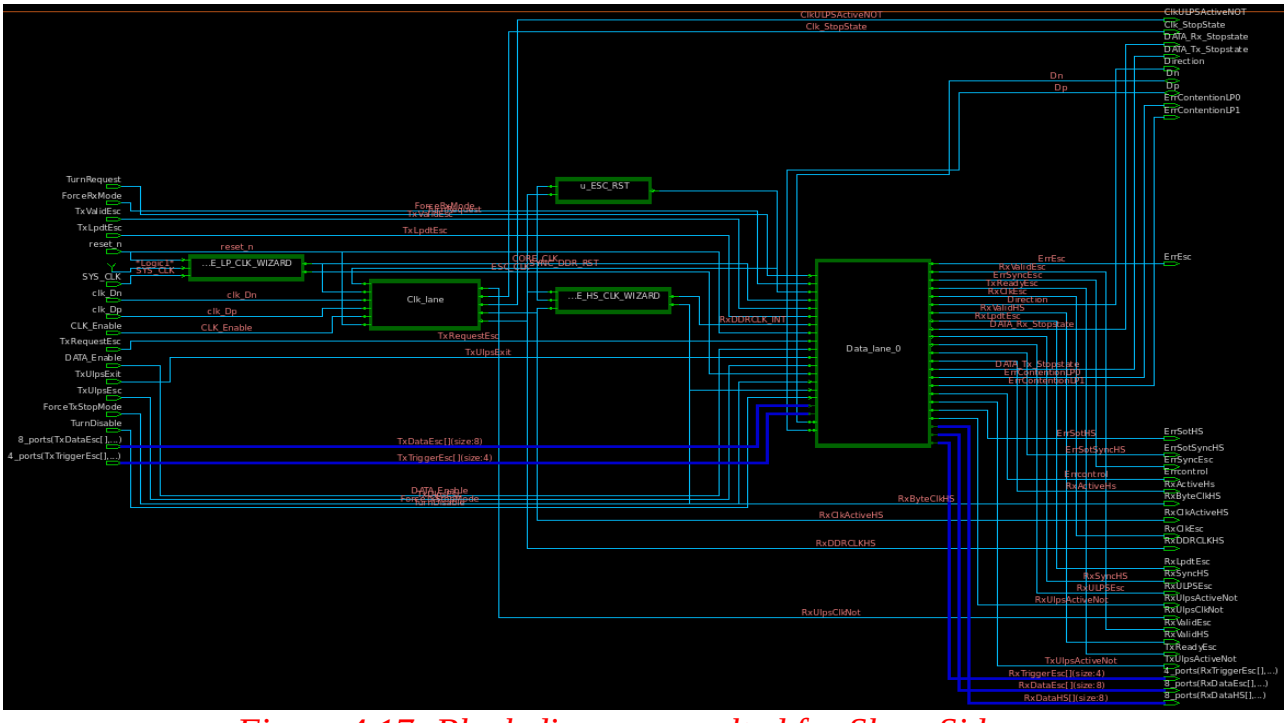


Figure 4.17: Block diagram resulted for Slave Side

4.2.2.3 Timing Reports Resulted

```
****************************************
Report : constraint
         -all_violators
Design : Slave_top
Version: K-2015.06
Date   : Sun May 11 02:06:24 2025
****************************************

This design has no violated constraints.


                    Design Rules
-----------------------------------
Total Number of Nets:          1724
Nets With Violations:             0
Max Trans Violations:             0
Max Cap Violations:               0
-----------------------------------
```

Figure 4.18: Constraints report for Slave Side

```
****************************************
Report : clocks
Design : Slave_top
Version: K-2015.06
Date   : Sun May 11 02:06:24 2025
****************************************

Attributes:
    d - dont_touch_network
    f - fix_hold
    p - propagated_clock
    G - generated_clock
    g - lib_generated_clock

Clock          Period   Waveform            Attrs     Sources
------------------------------------------------------------------------------
RxByteClkHS     20.00   {1.25 11.25}        G d       {SLAVE_HS_CLK_WIZARD/o_div_clk}
RxDDRClkHS       5.00   {1.25 3.75}         G d       {SLAVE_HS_CLK_WIZARD/o_ref_clk}
SLAVE_CORE_CLK
                10.00   {0 5}               G d       {SLAVE_LP_CLK_WIZARD/o_ref_clk}
SLAVE_ClkEsc    50.00   {0 25}              G d       {SLAVE_LP_CLK_WIZARD/o_div_clk}
SYS_CLK         10.00   {0 5}               d         {SYS_CLK}
clk_Dp           5.00   {1.25 3.75}         d         {clk_Dp}
ddr_q_demux      5.00   {1.25 3.75}         G d       {Clk_lane/U_SLAVE_CLK_DEMUX/HS_CLK}
ddr_q_rxhs       5.00   {1.25 3.75}         G d       {Clk_lane/U_Slave_HS/RxDDRClkHS}
------------------------------------------------------------------------------

Generated     Master         Generated      Master         Waveform
Clock         Source         Source         Clock          Modification
------------------------------------------------------------------------------
RxByteClkHS   Clk_lane/U_Slave_HS/RxDDRClkHS
                             {SLAVE_HS_CLK_WIZARD/o_div_clk}
                                            ddr_q_rxhs     divide_by(4)
RxDDRClkHS    Clk_lane/U_Slave_HS/RxDDRClkHS
                             {SLAVE_HS_CLK_WIZARD/o_ref_clk}
                                            ddr_q_rxhs     divide_by(1)
SLAVE_CORE_CLK
              SYS_CLK        {SLAVE_LP_CLK_WIZARD/o_ref_clk}
                                            SYS_CLK        divide_by(1)
SLAVE_ClkEsc  SYS_CLK        {SLAVE_LP_CLK_WIZARD/o_div_clk}
                                            SYS_CLK        divide_by(5)
ddr_q_demux   clk_Dp         {Clk_lane/U_SLAVE_CLK_DEMUX/HS_CLK}
                                            clk_Dp         divide_by(1)
ddr_q_rxhs    Clk_lane/U_SLAVE_CLK_DEMUX/HS_CLK
                             {Clk_lane/U_Slave_HS/RxDDRClkHS}
                                            ddr_q_demux    divide_by(1)
------------------------------------------------------------------------------
```

Figure 4.19: Clocks report for Slave Side

4.2.2.4 Power Reports Resulted

```
--------------------------------------------------------------------------------
                                       Switch   Int      Leak     Total
Hierarchy                              Power    Power    Power    Power    %
--------------------------------------------------------------------------------
Slave_top                              8.16e-02    0.506 1.02e+07    0.598 100.0
```

Figure 4.20: Power report for Slave Side

4.2.2.5 Area Reports Resulted

```
Area
-----------------------------------
Combinational Area:     9361.825389
Noncombinational Area:  9486.555311
Buf/Inv Area:            898.998823
Total Buffer Area:          118.85
Total Inverter Area:        780.15
Macro/Black Box Area:     0.000000
Net Area:                 0.000000
-----------------------------------
Cell Area:              18848.380701
Design Area:            18848.380701
```

```
Cell Count
-----------------------------------
Hierarchical Cell Count:         53
Hierarchical Port Count:        834
Leaf Cell Count:               1626
Buf/Inv Cell Count:             245
Buf Cell Count:                  25
Inv Cell Count:                 220
CT Buf/Inv Cell Count:            0
Combinational Cell Count:      1267
Sequential Cell Count:          359
Macro Count:                      0
-----------------------------------
```

Figure 4.21: Area report for Slave Side

4.3 LINTING RESULTS

We used to do Linting using SpyGlass.

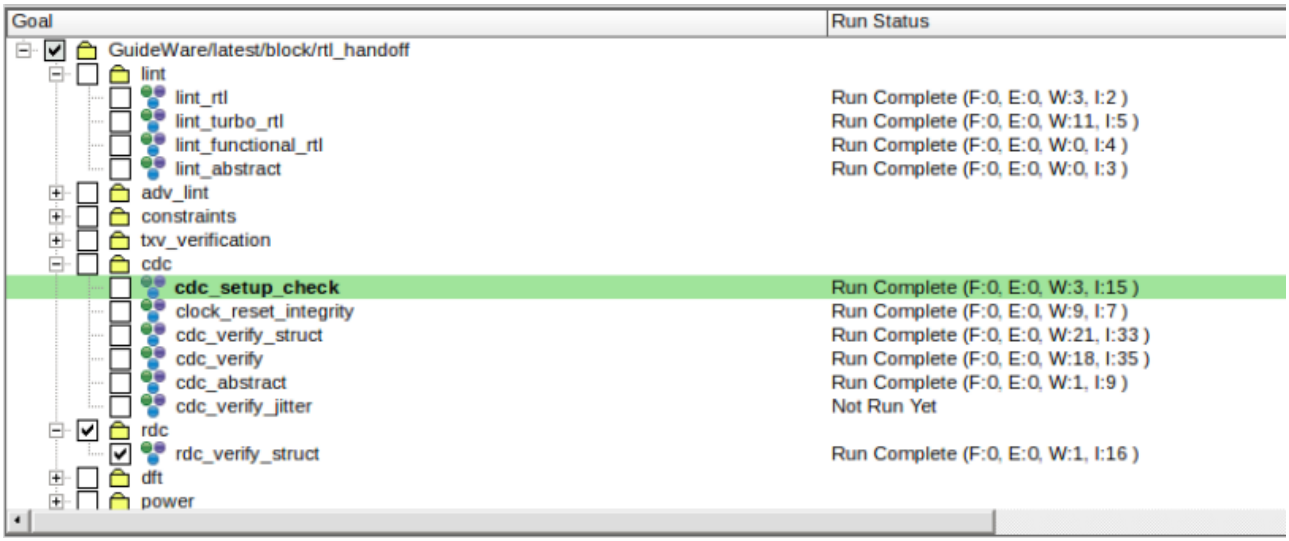


Figure 4.22: Result of Linting

4.4 IP RESULTS

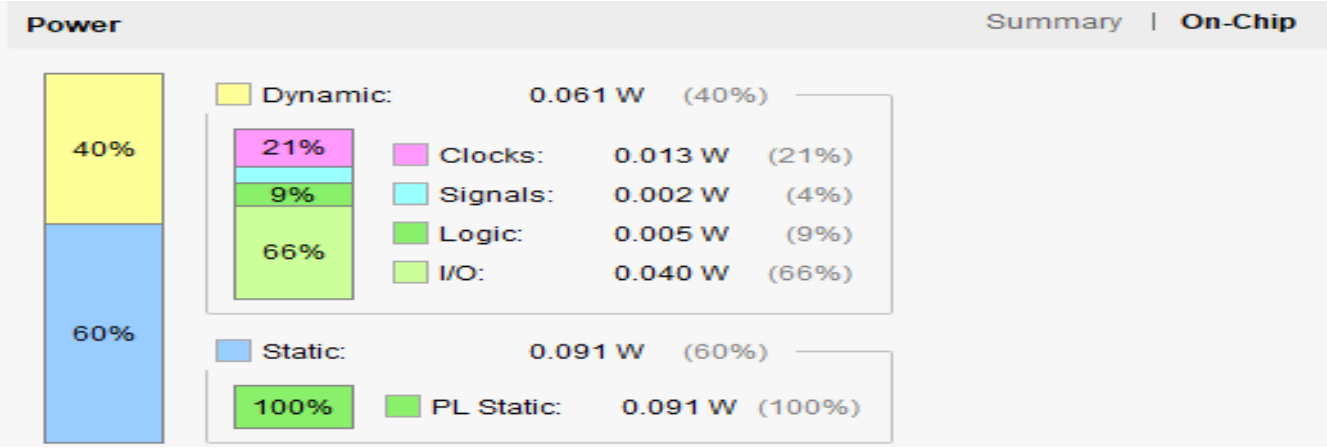
4.4.1 Result Timing for MIPI-D PHY IP using VIVADO

Design Timing Summary

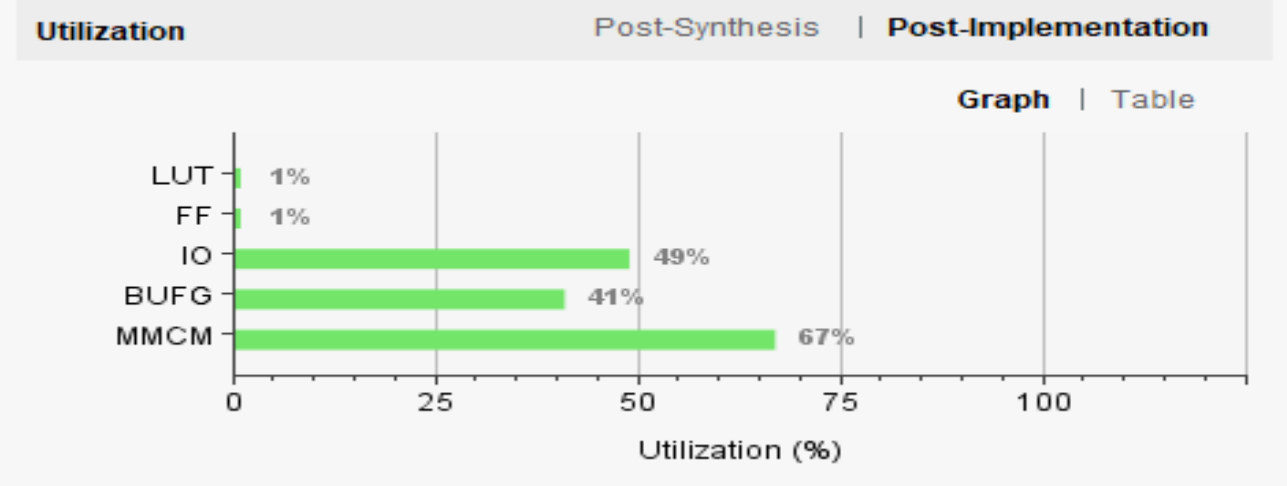
Setup		Hold	
Worst Negative Slack (WNS):	0.303 ns	Worst Hold Slack (WHS):	0.063 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	1339	Total Number of Endpoints:	1339

All user specified timing constraints are met.

4.4.2 Power Reports Resulted of IP



4.4.3 Utilization of IP



4.5 FPGA IMPLMENTATION

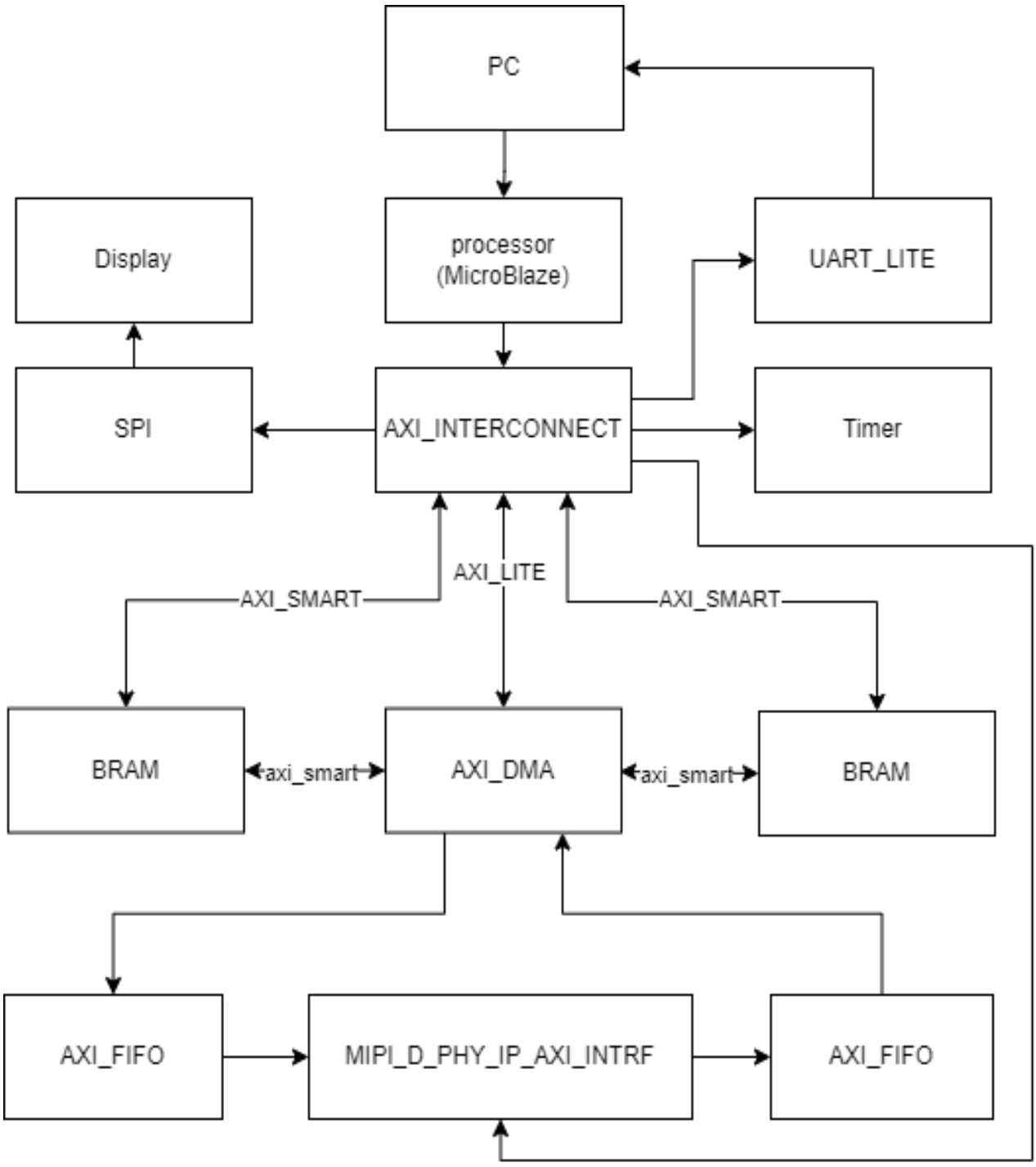


Figure 4.23: MIPI D PHY SOC Diagram

This system is designed to **test, validate, and interact with a MIPI D-PHY IP core**. It uses a **processor-based test platform** to send or receive data through the **MIPI_D_PHY_IP_AXI_INTRF** block, allowing various components to support configuration, data generation, transfer, and observation.

4.5.1 Blocks Explanation

PC

- External test controller or monitoring tool.
- Sends test commands or receives data for analysis.

Processor

- MicroBlaze is a 32-bit RISC soft processor by Xilinx, designed for FPGA-based systems.
- Central control for the entire test flow.
- Generates data patterns, configures DMA, and interacts with the PC.
- Interacts with MIPI D-PHY via AXI_LITE for control and AXI_SMART for data flow.

UART_LITE

- Interface to PC for control and status monitoring.
- Used to receive test results.

AXI Interconnect

- AXI Interconnect or AXI SmartConnect is an interconnect module that dynamically manages multiple AXI interfaces with high efficiency. It is used to connect AXI Masters (such as DMA) to AXI Slaves (such as DDR memory or custom IPs).

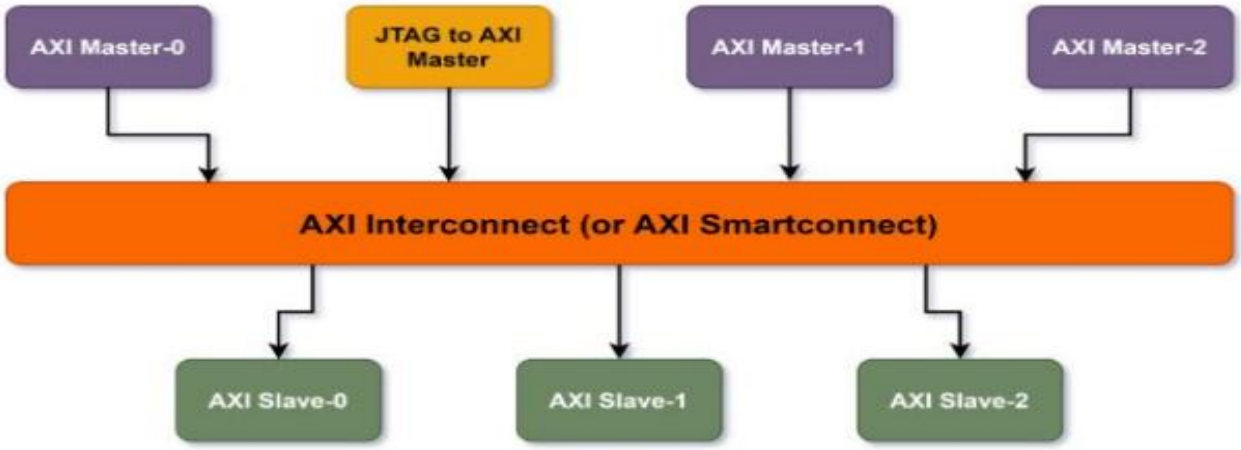


Figure 4.24: AXI Interconnect

AXI Stream

- AXI-Stream is a data streaming interface optimized for continuous, high-speed data flow. Unlike AXI Memory-Mapped, AXI-Stream does not use addresses, instead, it allows packet-based communication, making it ideal for processing pipelines**.**

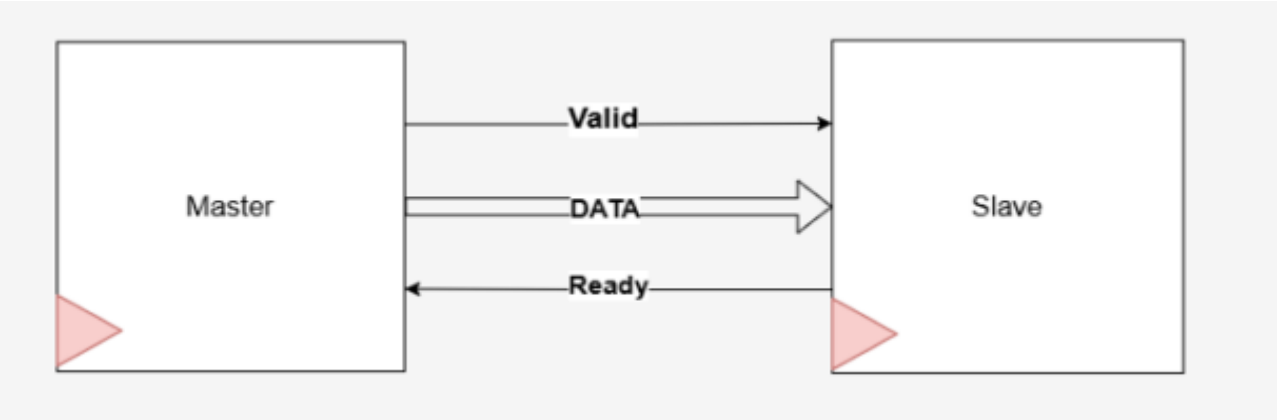


Figure 4.25: AXI Stream IP Core

DMA

- AXI DMA is a direct memory transfer engine that moves large amounts of data between BRAM memory and AXI-Stream peripherals without CPU intervention.
- It significantly improves performance by allowing direct data transfe

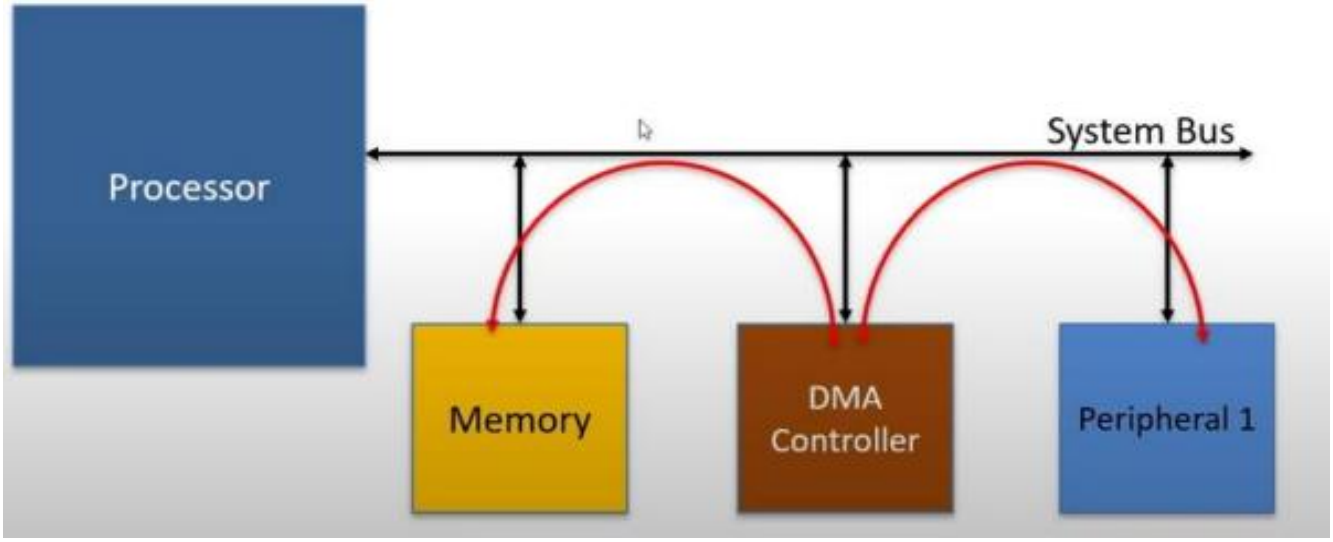


Figure 4.26: Direct memory access

BRAM

- Temporary storage for data to be transmitted or received.
- Acts as source/destination for DMA transactions.

AXI_FIFO

- Buffers data between AXI (100MHz) and the MIPI IP core (200MHz).
- Critical for handling burst data or clock domain crossing.
- Ensures stable data feeding to/from MIPI IP.

MIPI D PHY IP AXI INTRF

- Core under test.
- Implements a MIPI D-PHY interface.
- Connected to AXI and FIFOs to support integration with the rest of the system.

Timer

- Used for test timing, timeout monitoring, or triggering events.
- Useful for measuring latency or timing between Display initialization events.

SPI + Display

- Provides visual representation of the transmitted data by displaying the received image or test result on the screen.

4.5.2 System Scenarios for MIPI D-PHY Validation

This section presents two key operational scenarios that demonstrate the capabilities of the developed test system. Both scenarios leverage the MIPI D-PHY IP core integrated into a processor-based platform to validate its functionality using real image data.

Scenario 1: Automated Transmission, Reception, and Verification using Processor

In this scenario, the processor plays the role of a self-contained test controller. The purpose is to **transmit data through the MIPI D-PHY interface, receive it back via a loopback or external return path, and verify the integrity of the received data** without external user interaction.

Flow:

1. The processor initializes the system, including DMA, BRAM, FIFO, and MIPI D-PHY IP.
2. A known data pattern is generated or loaded into the processor memory.
3. The processor writes the test data into BRAM.
4. AXI_DMA is configured by the processor to transfer data from BRAM to AXI_FIFO.
5. The AXI_FIFO feeds data into the **MIPI D-PHY transmitter (Tx)**.
6. The data is received by another instance of the MIPI D-PHY receiver (Rx).
7. The Rx path writes data back into another FIFO, which is then moved into BRAM via DMA.
8. The processor reads the received data from BRAM and **compares it with the original data**.
9. Based on the result:
    - If data matches: a **"pass" result** is sent to the PC via UART.
    - If data mismatches: a **"fail" result** is sent to the PC.

Purpose:

- Automated self-check without external user logic.
- Used to **validate the complete MIPI transmission and reception chain**.
- Ensures system robustness before moving to visual output.

- **LPDT**

```
 UART_send_massege("  STARTING TEST IN FORWARD DIRECTION  ...\n\r");
 UART_send_massege("\n\r");
 UART_send_massege("\n\r");
 UART_send_massege("  STARTINGING LPDT TEST ...\n\r");
 UART_send_massege("  WE WILL SEND 100 BYTE OF DATA ...\n\n\r");

/* STARTINGINGing LPDT test*/
D_phy_MEsc_req(LPDT_REQ,0);

for (int i = 0; i < 100; i++){
 D_phy_Msend(i);
 if( D_phy_Srec() == i) correct_counter++;
 else error_counter++;
 }
D_phy_MEsc_dereq(); // assign 0 to each bit related to ESC_mode


if(correct_counter > 99){
    UART_send_massege("  LPDT TEST COMPLETED SUCCESSFULLY ...\n\r");
}
else if(correct_counter < 10){
    UART_send_massege("  LPDT TOTALLY FAILED ...\n\r");
 }
 else{
     UART_send_massege("  LPDT PARTIALLY PASSED ...\n\r");
 }
```

Figure 4.27: LPDT Testing code

- **Result of Initialization and LPDT**

```
SET CONFIGRATION FOR UART ...
SET CONFIGRATION FOR DMA ...
SET CONFIGRATION FOR GPIO ...

WRITING IMAGE TO FIRST BRAM ...
INTIALIZING AND RESETING THE SYSTEM D_phy ...
SYSTEM INITIALIZATION DONE ...


STARTING TEST IN FORWARD DIRECTION  ...


STARTINGING LPDT TEST ...
WE WILL SEND 100 BYTE OF DATA ...

LPDT TEST COMPLETED SUCCESSFULLY ...
NUMBER OF CORRECT DATA = 100
NUMBER OF FAILED DATA = 0
```

Figure 4.28: Initialization and LPDT Result

- **Triggers**

```
UART_send_massege("  STARTING TRIGGER TEST ...\n\r");

UART_send_massege("\n\r");
UART_send_massege("  SEND TRIGGER 1  ...\n\r");
   D_phy_MEsc_req(TRIGG_REQ,1);

   while(!((D_PHY->ESC_TRIGG_SLAVE) == 1));


   if( (D_PHY->ESC_TRIGG_SLAVE ) == 1)
       UART_send_massege("  TRIGGER 1 PASSED ...\n\r");
   else
       UART_send_massege("  TRIGGER 1 FAILED ...\n\r");

   D_phy_MEsc_dereq(); // assign 0 to each bit related to ESC_mode

   for (volatile u8 i = 0; i < 100; i++) {
    continue;
     }
```

Figure 4.29: Trigger Testing Code

- **Result of Trigger**

```
STARTING TRIGGER TEST  ...

SEND TRIGGER 1   ...
TRIGGER 1 PASSED ...

SEND TRIGGER 2   ...
TRIGGER 2 PASSED ...

SEND TRIGGER 3   ...
TRIGGER 3 PASSED ...

SEND TRIGGER 4   ...
TRIGGER 4 PASSED ...

TRIGGER TEST COMPLETLY DONE    ...
```

Figure 4.30: Result of trigger

- **ULPS**

```
 UART_send_massege("  STARTING ULPS TEST  ...\n\r");
 D_phy_MEsc_req(ULP_REQ,0);
 while(!GET_BIT(D_PHY->SR,RxULPSEsc_SLAVE));

if(!GET_BIT( D_PHY->SR,RxUlpsActiveNot_SLAVE))
 UART_send_massege("  ULPS PASSED SUCCESSFULLY ...\n\r");
else
 UART_send_massege("  ULPS FAILED ...\n\r");
D_phy_MEsc_dereq(); // assign 0 to each bit related to ESC_mode

UART_send_massege("  EXIT ULPS  ...\n\r");
D_phy_Esc_DATA_TxUlpsExit();
```

Figure 4.31: Code of ULPS

- **Result of ULPS**

```
STARTING ULPS TEST  ...
ULPS PASSED SUCCESSFULLY ...
EXIT ULPS  ...
```

Figure 4.32: Result of ULPS

- **TurnAround**

```
UART_send_massege("  STARTING TURNAROUND TEST  ...\n\r");
D_phy_MTurn_req(TURN_1);
for (volatile u32 i = 0; i < 100; i++) {
     continue;
     }

D_phy_MTurn_req(TURN_0);

 UART_send_massege("\n\r");
if(!GET_BIT( D_PHY->SR,Direction_SLAVE) && GET_BIT( D_PHY->SR,Direction_MASTER))
    UART_send_massege("  TurnAround DONE SUCCEFULLY ...\n\r");
else
    UART_send_massege("  TurnAround FAILED ...\n\r");
```

Figure 4.33: Code of TurnAround

- **Result of TurnAround**

```
STARTING TURNAROUND TEST  ...

TurnAround DONE SUCCEFULLY ...
```

Figure 4.34: Result of TurnAround

- **Result of testing in reverse direction**

```
STARTING TEST IN REVERSE DIRECTION  ...


STARTING LPDT TEST ...
WE WILL SEND 100 BYTE OF DATA  ...

LPDT COMPLETED SUCCESSFULLY  ...
NUMBER OF CORRECT DATA = 100
NUMBER OF FAILED DATA = 0


STARTING TRIGGER TEST ...
SEND TRIGGER 1  ...
TRIGGER 1 PASSED ...

SEND TRIGGER 2  ...
TRIGGER 2 PASSED ...

SEND TRIGGER 3  ...
TRIGGER 3 PASSED ...

SEND TRIGGER 4  ...
TRIGGER 4 PASSED ...

TRIGGER TEST COMPLETLY DONE   ...


STARTING ULPS TEST  ...
ULPS PASSED SUCCESSFULLY ...
EXIT ULPS  ...


STARTING TURNAROUND TEST AGAIN ...

TurnAround DONE SUCCESSFULLY ...
```

Figure 4.35: Result of testing in reverse direction

Scenario 2: Sending a Real Image and displaying it on a Screen

In this scenario, the system demonstrates **practical data transmission over MIPI D-PHY by sending a real image from the processor and displaying it on a connected screen**.

Flow:

1. To load image data into **BRAM** for processing, we will use **HxD**, a hex editor tool, to load image data into **BRAM** for processing. It allowed us to **convert image files to binary or hex format**.
2. The processor receives or stores a pre-defined figure or image (bitmap or hex format).
3. The image is loaded into BRAM.
4. DMA is configured to transfer image data from BRAM to AXI_FIFO.
5. AXI_FIFO buffers the image data and feeds it into the **MIPI D-PHY transmitter**.
6. The Rx path writes data back into another FIFO, which is then moved into BRAM via DMA.
    - The processor reads the received data from BRAM and send it to a connected **SPI interface**.
    - Sent via SPI to a **display device**.
7. The image is rendered on the display, showing that **data was transmitted correctly through the MIPI D-PHY interface**.

Purpose:

- Provides a **visual confirmation** of successful transmission.
- Simulates a real-world use case such as camera-to-display pipelines.
- Demonstrates system throughput and synchronization under realistic data loads.

- **Original image**

Figure 4.36: Original image

- **TFT Image**

Figure 4.37: TFT image

CHAPTER 5: REFERENCES

Books:

[1] MIPI Alliance Specification for D-PHY Version 2.1

[2] LRM: IEEE Standard for SystemVerilog—Unified Hardware Design, Specification, and Verification Language

[3] SystemVerilog for Verification A Guide to Learning the Testbench Language Features by Chris Spear, Greg Tumbush

[4] Python Crash Course, 2nd Edition: A Hands-On, Project-Based Introduction to Programming

[5] UVM Cookbook, by Verification Academy, Sponsered by Mentor Graphics

[6] Python for RTL Verification A complete course in Python, cocotb, and pyuvm

Official Websites:

[7] UVM | Verification Academy

[8] WWW.TESTBENCH.IN

[9] SystemVerilog Tutorial (chipverify.com)

[10] SystemVerilog Tutorial (asic-world.com)

[11] Python Tutorial (w3schools.com)

[12] Learn Python 3 | Codecademy

[13] uvm_guide.book (accellera.org)

[14] UVM Tutorial - Verification Guide

[15] UVM Tutorial for Beginners (chipverify.com)

ملخص المشروع

يتناول هذا المشروع تصميم والتحقق من بروتوكول **MIPI D PHY**

والذي يُستخدم كواجهة نقل بيانات عالية السرعة ومنخفضة الطاقة في الأجهزة المحمولة، مثل الهواتف الذكية والكاميرات

بدأنا المشروع باستخلاص المواصفات التقنية،

ثم قمنا بتصميم الوحدات الرئيسية لبروتوكول D-PHY مثل

(Transmitter - Tx) وحدة الإرسال
(Receiver - Rx) وحدة الاستقبال
(Clock Lane) مسار الساعة

يدعم التصميم وضعين للتشغيل

(HS) الوضع عالي السرعة لنقل البيانات السريع،

(LP) الوضع منخفض الطاقة لنقل البيانات البسيطة والتحكم.

بعد الانتهاء من التصميم، قمنا بإنشاء بيئة تحقق باستخدام

UVM (Universal Verification Methodology)

حيث تم بناء بنية تحقق متكاملة تشمل

(Sequences) سينسرات
(Drivers) درايفرات
(Monitors) مونيتورات
(Scoreboards & Coverage) وحدات التحقق والنتائج

تم إجراء محاكاة شاملة للتحقق من صحة التصميم، وأظهرت النتائج أن النظام يعمل كما هو متوقع وفقًا للمواصفات. يبرهن المشروع على قدرة الفريق في تصميم وتنفيذ واجهات رقمية معقدة والتحقق منها بطريقة احترافية.

جامعة الازهر
كلية الهندسة
قسم الهندسة الكهربية
شعبة الاتصالات والالكترونيات

مشروع التخرج بعنوان

التصميم والتحقق من بروتوكول نقل الصور ***MIPI D PHY***

وتنفيذه بواسطة ***FPGA***

تم اعداده بواسطة

احمد محمد علي احمد المصري	**١٠٤٥٠٢**
احمد محمد حامد الملاح	**١٠٤٥٠٣**
السيد طلعت السيد تمام	**١٠٤٥٠٤**
محمد امام حسين عبدالسلام	**١٠٤٥١٣**
محمد منصور عثمان احمد	**١٠٤٥١٩**
يوسف عبد الفتاح إبراهيم يونس	**١٠٤٥٢٦**

تحت اشراف

د.عبدالله حسان

برعاية شركة ساي فيجن

Si-VISION
INNOVATIVE IC SOLUTIONS

يوليو ٢٠٢٥